**CURSO 2014 – 2015**

# Aplicación de la metodología GloBeM al servicio de cloud Hydra

## Trabajo Fin de Máster

**Autor: Javier Ferrero Merchán**

**Tutor: María de los Santos Pérez Hernández**

**Cotutor: Jesús Montes Sánchez**

## Resumen

El presente Proyecto de Fin de Máster consiste en crear una herramienta software capaz de monitorizar y gestionar la actividad de *Hydra*, una herramienta de gestión de entornos distribuidos, para que su estrategia de balanceo de carga se adecúe al modelo creado por *GloBeM*, una metodología de análisis de entornos distribuidos.

GloBeM, que es una metodología externa, puede analizar y crear un modelo de máquina de estados finitos a partir de un sistema distribuido concreto.

Hydra, una herramienta también externa, es un sistema de gestión de entornos cloud recientemente desarrollado y de código abierto, con un sistema de balanceo de carga efectivo pero algo limitado.

El software construido recoge el modelo creado por GloBeM y lo analiza. A partir de ahí, monitoriza en tiempo real y a una frecuencia determinada la actividad de Hydra y el sistema cloud que ésta gestiona, y reconfigura sus parámetros para que su desempeño se ciña a lo estipulado por el modelo de GloBeM, extendiendo así el sistema de balanceo de carga original de Hydra.

## Abstract

This Master's Thesis Project involves creating a software able to monitor and manage the activity of *Hydra*, a tool for managing distributed environments, in order to adjust its load balancing strategy to the model created by *GloBeM*, an analysis methodology for distributed environments.

GloBeM, which is an external methodology, can analyse and create a finite-state machine model from a particular cloud system.

Hydra, also an external tool, is an open source management system for cloud environments recently developed, with a relatively limited system of load balancing.

The created software gets the model created by GloBeM as an input and analyses it. From there, it monitors in real time and at a certain frequency Hydra’s activity and the cloud system that it manages, and reconfigures its parameters to adjust its performance to the stipulations by the GloBeM’s model, extending Hydra's original load balancing system.

# Tabla de contenido

# Tabla de ilustraciones

# 1 Introducción

Dentro del mundo de las tecnologías de la información, en la actual era digital, la presencia y uso generalizado de las redes de comunicaciones para muy diversos fines es, desde hace tiempo, más que una realidad: poco a poco se convierte en una necesidad imperativa.

La cantidad de fines diversos que se les pueden dar a las redes, con la posibilidad que ofrecen de interconectar equipos entre sí, está limitada por poco más que la imaginación. Una de estas posibles aplicaciones es la capacidad de formar conglomerados de equipos informáticos que trabajen en conjunto para realizar tareas en común, los llamados *clústeres*.

Esta idea tan simple (de que la unión hace la fuerza) ha dado origen a multitud de modelos, arquitecturas o estructuras de comunicaciones e interconexión de equipos que permiten expandir enormemente las posibilidades informáticas, en términos de almacenamiento de datos, procesamiento, seguridad, etc., desbancando casi por completo el uso de los antiguos superordenadores, cuyos costes de construcción y mantenimiento han demostrado ser mucho menos rentables ante la opción de la creación de clústeres informáticos.

Estos clústeres de equipos creados en torno a una infraestructura de telecomunicaciones distribuida, permiten compartir grandes cargas de trabajo entre sus nodos, aprovechando el enorme poder de procesamiento y almacenaje que acumulan entre todos ellos. Y para que dichos recursos informáticos se aprovechen de manera eficaz y eficiente, es necesario el uso de sistemas de gestión de dichos entornos.

La mayoría de sistemas de gestión de entornos distribuidos actuales se basan en el análisis y control de todos y cada uno de los componentes que lo forman. La exhaustividad de este enfoque proporciona múltiples beneficios, pero a menudo la enorme complejidad de los entornos distribuidos hace muy difícil su aplicación y correcto funcionamiento.

Como alternativa, existe la posibilidad de analizar el comportamiento del sistema distribuido como si de un único sistema de tratase, adoptando un enfoque de alto nivel basado en los servicios que el entorno proporciona. Este enfoque permite tener en cuenta aspectos globales relativos al funcionamiento del entorno y las sinergias que pueden tener lugar entre los múltiples recursos que lo componen.

Para tal fin, miembros de la Universidad Politécnica de Madrid y la Universidad Rey Juan Carlos (de las escuelas de informática de ambos), definieron el llamado sistema GloBeM (Global Behavior Modeling) [1], una metodología de análisis especialmente diseñada con el objetivo de construir un modelo de comportamiento global de entornos distribuidos.

GloBeM es un sistema complejo, cuyo desempeño se divide en varias fases, cada una de las cuales proporciona una salida resultado, que se convierte en los datos de entrada para la siguiente. El resultado final de esta metodología es un modelo que permite comprender el comportamiento del sistema distribuido que ha sido analizado.

El objetivo principal del proyecto al que hace referencia el presente documento es el de crear una herramienta capaz de recoger el resultado final de la aplicación de GloBeM sobre un sistema distribuido, y añadir la comprensión de este modelo a otro sistema de gestión de entornos distribuidos para mejorar su labor, concretamente en cuanto al balanceo de carga [2], [3], [4] se refiere (una de las cuestiones más complejas a las que estos sistemas de gestión deben hacer frente). El sistema de gestión elegido con el que trabajará la herramienta construida, es el sistema Hydra [5].

# 2 Visión general

## 2.1 Propósito y contenido

El propósito de este Proyecto de Fin de Máster es el de analizar, diseñar, construir y validar una herramienta software que haga de intermediario entre el sistema GloBeM e Hydra, que es la herramienta software de gestión de entornos cloud que en este proyecto se va a utilizar para ser mejorada con el uso de GloBeM. Entre sus funciones estarán las de recoger y entender el modelo creado por GloBeM, así como monitorizar y reconfigurar Hydra para lograr que su labor se adecúe a tal modelo. Y para ello, se dispone inicialmente de una descripción informal de los requisitos de la herramienta, además de documentación de apoyo sobre los sistemas GloBeM e Hydra, y la tutorización por parte de alguno de los miembros que ayudaron a definir el sistema GloBeM.

Este documento va a cubrir todo el proceso del ciclo de vida del software, empezando por una extracción y análisis formal de los requisitos del sistema, el diseño de alto y bajo nivel (arquitectura y vistas del sistema) y la documentación de las pruebas de verificación y validación del software. La parte propiamente de la construcción del software (la codificación) no se va a documentar, puesto que no se considera necesario, ni se va a exponer el código entre estas páginas (ya que es excesivamente extenso).

En cualquier caso, la documentación de todo esto se realizará desde la perspectiva de dar una visión tanto del producto final obtenido, como del trabajo que se ha realizado para conseguirlo.

Además de todo esto, se incluyen dos secciones más en la memoria. La primera, al principio, se va a centrar en dar una visión general de la situación actual de las tecnologías que tienen relación con el actual proyecto, es decir, tipos y usos de sistemas distribuidos, otras herramientas software de gestión de estos entornos, etc., así como explicar en detalle los sistemas GloBeM e Hydra.

La segunda sección adicional está al final del documento, y está destinada a exponer las conclusiones que el autor del proyecto puede sacar de su realización, así como líneas futuras, es decir, el trabajo de evolución de la herramienta que posiblemente podría ser realizado en un futuro a partir de la situación en que ha quedado tras la finalización de este PFM.

## 2.2 Ámbito

El actual Proyecto de Fin de Máster se va a centrar en la creación de una herramienta para la monitorización y gestión en tiempo real del sistema Hydra, usando como base de su actuación un modelo generado por el sistema GloBeM.

Esto implica que la herramienta pueda reconocer adecuadamente cualquier modelo generado por GloBeM, obtenido al analizar un sistema distribuido cualquiera, pero su aplicación sólo será posible para el sistema de gestión Hydra. Para ser aplicado a otro sistema, sería necesario adaptar la herramienta.

# 3 Glosario

**Arquitectura software.** Se trata de la definición a más alto nivel del diseño de un sistema software, en términos de sus componentes computacionales y sus elementos software, las propiedades visibles externamente de esos elementos, y las relaciones e interacciones entre ellos.

**Clúster.** Del inglés clúster ("grupo" o "racimo"), se refiere a un conjunto de computadoras creado mediante la utilización de hardware común y que se comporta como si de una sola computadora se tratara.

**Computación en la nube.** Sistema informático basado en Internet y centros de datos remotos para gestionar servicios de información y aplicaciones. Permite que los consumidores y las empresas gestionen archivos y utilicen aplicaciones sin necesidad de instalarlas en cualquier computadora con acceso a Internet.

**Computación grid.** Un modelo de trabajo para compartir recursos de todo tipo (entre ellos, de cómputo, almacenamiento y aplicaciones específicas) en forma coordinada y sin un control centralizado.

**Computación monolítica.** La forma de computación más sencilla, en la que sólo se hace uso de un único ordenador (una sola CPU), el cual no está conectado a ninguna red y, por tanto, sólo puede utilizar aquellos recursos a los que tiene acceso de manera inmediata. Puede ser mono o multi usuario.

**Cuestión de diseño.** Un área de interés con respecto a un diseño software.

**Data mining (minería de datos).** Campo de las ciencias de la computación referido al proceso de intentar descubrir patrones en grandes volúmenes de conjuntos de datos, con el fin de extraer información oculta no trivialmente identificable.

**etcd.** Un almacén de pares clave-valor de alta disponibilidad para la configuración compartida y el descubrimiento de servicios en entornos distribuidos.

**GloBeM.** (Global Behavior Modeling) Una metodología de análisis especialmente diseñada con el objetivo de construir un modelo de comportamiento global de entornos distribuidos.

**Hydra.** Un servicio de descubrimiento, gestión y balanceo de aplicaciones multi-cloud, el cual, pretende facilitar el enrutamiento y el balanceo de carga de los servidores y delegarla en el cliente (navegador, aplicación móvil, etc.).

**OpenStack.** Software de código abierto diseñado para controlar grandes conjuntos de recursos de computación, almacenamiento y red, gestionados a través de un panel de control para crear un entorno de trabajo distribuido.

**Punto de vista de diseño.** La especificación de los elementos y convenciones disponibles para construir y usar una vista de diseño.

**Sistema distribuido.** Un sistema cuyos componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor.

**Stakeholder.** Un individuo, organización o grupo (o alguna clase de los mismos jugando el mismo rol) que tenga un interés en, o cuestiones de diseño en relación con, el diseño de algún elemento del software.

**Sujeto de diseño.** Un elemento software o sistema para el que se va a preparar un documento de especificación de requisitos.

**Vista de diseño.** Una representación compuesta de uno o más elementos de diseño para hacer frente a un conjunto de cuestiones de diseño desde un punto de vista de diseño específico.

*Parte I*

# Estado de la cuestión

*Computación distribuida*

*El sistema GloBeM*

*Hydra*

# 4 Computación distribuida

Hasta la invención de las redes de comunicaciones, los equipos informáticos funcionaban siempre de forma aislada y dependiendo únicamente de sus propios recursos informáticos para poder funcionar. Es lo que se conoce como computación monolítica. Esto por sí solo supuso una revolución en cuanto a las posibilidades de realización de cálculos y procesamiento de datos. Pero a medida que esta tecnología avanzaba, también lo hacían las necesidades de potencia de procesamiento. Cuando era necesario un gran poder de cálculo o rendimiento, se recurría a los antiguos superordenadores, también llamados *mainframes*, en general de mucha potencia y rendimiento, pero de altísimo coste.

Gracias al desarrollo y evolución de las redes de comunicaciones, las computadoras pudieron conectarse y empezar a compartir recursos (inicialmente datos, como el correo electrónico). Y no se tardó mucho en ver las posibilidades que tenían de compartir recursos de computación, consiguiendo así que los equipos informáticos trabajasen en común para lograr un mismo fin.

De este modo, nació la computación distribuida [6], un modelo para resolver problemas de computación masiva utilizando un gran número de ordenadores, los cuales se organizaban en clústeres y se incluían dentro de una infraestructura de telecomunicaciones distribuida.

Así, definimos un sistema distribuido como aquel formado por dos o más componentes, localizados en computadoras independientes no sincronizadas, interconectadas a través de una red de comunicaciones de cualquier tipo, y que se comunican y coordinan mediante el paso de mensajes para realizar una tarea en común. Los equipos que forman el sistema pueden encontrarse en cualquier parte, ya sea en un mismo rack, o en la otra punta del mundo; y ante el usuario, suele resultar deseable que se muestren de forma transparente como si de un solo ordenador se tratara.

El enorme éxito que ha tenido la computación distribuida frente a la monolítica se debe fundamentalmente a las siguientes razones:

- **Los costes**. El uso de computadores de potencia media y de una red de comunicaciones que los interconecte resulta mucho más barato y rentable que comprar y mantener un superordenador, obteniendo la misma potencia o incluso más que éste.
- **Compartición de recursos**. El acceso a través de redes de comunicaciones a los equipos que conforman el sistema distribuido que es el sistema información de las empresas y las organizaciones, permite que se puedan compartir recursos de todo tipo (desde documentos o aplicaciones, hasta recursos hardware) tanto fuera de la compañía como dentro.
- **Escalabilidad**. En la computación distribuida resulta mucho más sencillo variar el número de recursos de que se disponen en el sistema, ya sea añadiendo o quitando equipos, lo cual añade una gran capacidad de escalabilidad y

flexibilidad en el diseño de la misma. En cambio, los sistemas monolíticos se encuentran mucho más limitados por la configuración inicial con que se crearon.

- **Tolerancia a fallos**. Los sistemas distribuidos facilitan la replicación de recursos, gracias a lo cual pueden seguir funcionando aunque alguno de los equipos que forman el sistema haya dejado de funcionar.

A su vez, la computación distribuida también ha traído consigo nuevos retos o inconvenientes:

- **Seguridad**. En un sistema distribuido hay más oportunidades de ataques no autorizados. Mientras que en un sistema centralizado todos los computadores y recursos están normalmente bajo el control de una administración única, algunos tipos de sistemas distribuidos (como los grids y aplicaciones similares) suelen implicar una gestión descentralizada y a un gran número de organizaciones independientes. La descentralización hace difícil implementar y ejecutar políticas de seguridad; por tanto, la computación distribuida en estos casos es vulnerable a fallos de seguridad y accesos no autorizados, que desafortunadamente puede afectar a todos los participantes en el sistema.
- **Múltiples puntos de fallo**. Existen más puntos de fallo en la computación distribuida, debido a que implica múltiples computadoras y todas son dependientes de la red para su comunicación. Así, el fallo de una o más computadoras, o uno o más enlaces de red, puede suponer problemas para un sistema de computación distribuida.

## 4.1 Computación grid y en la nube

La computación grid y en la nube son dos modelos de computación distribuida muy parecidas entre sí, con la característica en común de que son sistemas que buscan compartir el acceso a sus recursos, ya sea para realizar tareas en común, o para dar servicios a través de Internet a un gran número de usuario y con requisitos altos de disponibilidad, tolerancia a fallos, potencia, etc.

Una grid es una colección de computadoras, usualmente pertenecientes a múltiples agrupaciones en diversos lugares, interconectadas de forma tal que los usuarios pueden compartir el acceso a su poder combinado.

Una nube es una colección de computadoras, usualmente de propiedad de un único grupo, interconectadas de modo tal que los usuarios pueden arrendar el acceso a ellas para compartir el poder combinado que ofrecen.

Mientras que las grids están más orientadas a compartir los servicios y recursos de que disponen de manera gratuita o con fines colaborativos de tipo, por ejemplo, científico, las nubes están más orientadas a ofrecer una serie de servicios por parte de una organización de TI a otras organizaciones a través de un pago según el uso.

En cualquier caso, en ambos sistemas se requiere el uso de algoritmos que permitan utilizar coordinadamente todos los equipos disponibles en la red. Entre otros, unos de los mayores retos que deben afrontar estos sistemas es el *balanceo de carga*.

## 4.2 Balanceo de carga

Cuando hablamos de balanceo de carga en el ámbito de sistemas distribuidos, nos referimos a la técnica empleada para compartir el trabajo a realizar entre varios equipos, que dispongan de los recursos necesarios y sean capaces de realizarlo.

Se realiza gracias a algoritmos de balanceo que dividen de la manera más precisa y equitativa posible el trabajo, encuentran el mapeo de tareas que resulte proporcional y consiguen que cada componente tenga una cantidad de trabajo que demande aproximadamente el mismo tiempo, para evitar los llamados cuellos de botella entre los elementos que lo componen.

Equilibrar la carga de trabajo de los procesadores incrementa la eficiencia global y reduce el tiempo de ejecución. La administración del balanceo es particularmente compleja si los procesadores (y las comunicaciones entre ellos) son heterogéneos, ya que deben tenerse en cuenta protocolos, velocidades, sistemas operativos, comunicaciones, etc.

Existen diversas formas de clasificar los algoritmos de balanceo de carga. Aquí, lo haremos atendiendo al momento en que se ejecutan dichos algoritmos:

- **El balanceo de carga estático**. También llamado mapeado del problema o planificación del problema. Este tipo de balanceo de carga se trata antes de la ejecución de cualquier proceso. El balanceo de carga estático tiene serios inconvenientes que lo sitúan en desventaja frente al balanceo de carga dinámico. Entre ellos, cabe destacar:
    - Es muy difícil estimar de forma precisa el tiempo de ejecución de todas las partes en las que se divide un programa sin ejecutarlas.
    - Algunos sistemas pueden tener retardos en las comunicaciones que pueden variar bajo diferentes circunstancias, lo que dificulta incorporar la variable retardo de comunicación en el balance de carga estático.
    - A veces los problemas necesitan un número indeterminado de pasos computacionales para alcanzar la solución. Por ejemplo, los algoritmos de búsqueda normalmente atraviesan un grafo buscando la solución, y a priori no se sabe cuántos caminos hay que probar, independientemente de que la programación sea secuencial o paralela.
- **El balanceo de carga dinámico**. Se realiza durante la ejecución de los procesos. Con este tipo de balanceo, todos los inconvenientes que presenta el balanceo de carga estático son atajados, puesto que la división de la carga computacional depende de las tareas que se están ejecutando y no de la estimación del tiempo que pueden tardar en ejecutarse. Aunque el balanceo de carga dinámico lleva

consigo una cierta sobrecarga durante la ejecución del programa, resulta una alternativa mucho más eficiente que el balanceo estático.

En el balanceo de carga dinámico, las tareas se reparten entre los procesadores durante la ejecución del programa. Dependiendo de dónde y cómo se almacenen y repartan las tareas, el balanceo de carga dinámico se divide en:

- *Balanceo de carga dinámico centralizado*. Se corresponde con la estructura típica de maestro/esclavo.
- *Balanceo de carga dinámico distribuido o descentralizado*. Se utilizan varios maestros y cada uno controla a un grupo de esclavos.

A continuación, se ejemplifica la evolución de los algoritmos de balanceo de carga dinámicos, con la exposición de algunos de los más representativos.

### *Round Robin*

Se trata de un algoritmo de balanceo donde las peticiones de clientes son distribuidas equitativamente entre todos los equipos existentes. Este método cíclico no tiene en cuenta las condiciones ni la carga de cada computador. Esto puede traer consigo tener equipos que reciben peticiones de carga hasta colapsar, mientras se tiene otros equipos que apenas se encuentran ejecutando tareas, debido principalmente, a que los sistemas distribuidos tienden a ser muy heterogéneos.

Round Robin fue uno de los primeros algoritmos diseñados para el balanceo de carga (por su sencillez).

### *Pasive Polling*

En este método, el balanceador de carga calcula el tiempo de respuesta de los equipos y tiene una referencia de su estado. No se tiene en cuenta la variedad de computadoras empleadas. Además, sólo descubre que los equipos tienen un problema después de que se produzcan retrasos o, en el peor de los casos, cuando los equipos están completamente caídos.

Este método mejora un poco el anterior, al realizar una monitorización en tiempo real del estado del sistema, e intentar reaccionar ante situaciones anómalas que se produzcan, aunque no lo haga de manera perfecta.

### *Nodo de balanceo*

Ésta es la metodología más moderna y actual de afrontar los problemas de balanceo de carga, y se basa en que las peticiones de servicio al sistema distribuido se realicen siempre a través de un único punto de entrada, un nodo que será el balanceador de carga, y el cual, de forma transparente, dirigirá el tráfico a cualquiera de los nodos disponibles. Para lograrlo, el balanceador de carga continuamente realiza peticiones de datos a cada equipo del grupo que monitoriza, y direcciona las peticiones hacia el equipo que se encuentre más disponible y en mejor estado para responder a dichas peticiones. Los parámetros solicitados, dependen del producto utilizado. Normalmente,

se emplea la utilización de la CPU de la computadora, el uso de la memoria y el número de conexiones abiertas.

Aunque se habla de un nodo que lleva a cabo la gestión del balanceo para el resto de nodos, no hay que entenderlo como un sistema centralizado, puesto que se pueden tener numerosos nodos que se coordinan para realizar esta función.

El sistema GloBeM, objeto de estudio en este proyecto, cumple con las características de los balanceadores descritos en este último grupo, y aunque puede realizar más funciones, en este proyecto se va a utilizar precisamente para el balanceo de cargas.

# 5 El sistema GloBeM

La gran mayoría de los sistemas de gestión de entornos distribuidos actuales que siguen el modelo de nodo de balanceo han ido evolucionando desde modelos anteriores, los cuales se aplicaban a sistemas distribuidos bastante más sencillos. Estos se basan, por tanto, en el análisis exhaustivo de los recursos que componen el sistema y en el ajuste independiente de los parámetros que gobiernan el comportamiento de cada uno de ellos.

Sin embargo, el desarrollo de las tecnologías de computación distribuida ha añadido gran complejidad a estos sistemas. Al intentar adaptar este enfoque a sistemas muy grandes como grids o clouds, el inmenso tamaño y complejidad del entorno dificulta enormemente el proceso.

GloBeM surgió con la intención de dar un nuevo punto de vista a este problema. Se basa en la posibilidad de modelar el comportamiento del entorno distribuido como si de una única entidad se tratase, proporcionando una mayor y mejor comprensión de su funcionamiento y de las sinergias que tienen lugar entre sus múltiples componentes. Este conocimiento puede aplicarse entonces a desarrollar nuevas aplicaciones y eficientes técnicas de gestión, teniendo en cuenta toda la nueva información adquirida.

A modo de ejemplo, un ordenador personal es una situación parecida. Está compuesto por una gran cantidad de elementos hardware individuales muy complejos, que mantienen relaciones estrechas entre todos ellos para funcionar. Comprender su funcionamiento a nivel físico de manera exhaustiva resulta extremadamente difícil, pero en realidad, para el usuario, esto no es necesario, puesto que dispone de una serie de herramientas de alto nivel (como el sistema operativo y las aplicaciones) que le permiten aprovechar todo el potencial que la máquina ofrece. Este mismo concepto es el que GloBeM traslada a la gestión de entornos distribuidos.

Así, el objetivo principal de GloBeM es construir un modelo de comportamiento global de un entorno distribuido. Los modelos generados cumplirán ciertas características para ser representativos y relevantes:

- **Definición específica de estados**. Las características de los distintos estados del sistema posibles y sus transiciones deben estar correctamente representadas. Una

representación comúnmente usada que cumple este requisito es una *máquina de estados finitos*.

- **Estabilidad**: El modelo debe representar el comportamiento del sistema a lo largo del tiempo. La variabilidad de estos entornos dificulta generar un modelo definitivo para cualquier instante, pero se deben poder generar modelos con un nivel aceptable de estabilidad en el tiempo.
- **Sencillez**: Los administradores y desarrolladores de aplicaciones deben poder utilizar el modelo, por lo que debe ser lo suficientemente simple para ser usable.
- **Relevancia**: La información proporcionada por el modelo debe ser representativa del comportamiento global del sistema.

El resultado final de la aplicación de GloBeM es una máquina de estados finitos, que se convierte en una capa de abstracción del entorno, la cual expresa el comportamiento del sistema distribuido de forma simple, relevante y usable. Esta representación hace posible la visión del sistema como una única entidad.

El proceso se estructura en las siguientes fases:

1. Observación del sistema
2. Análisis de datos
3. Construcción del modelo

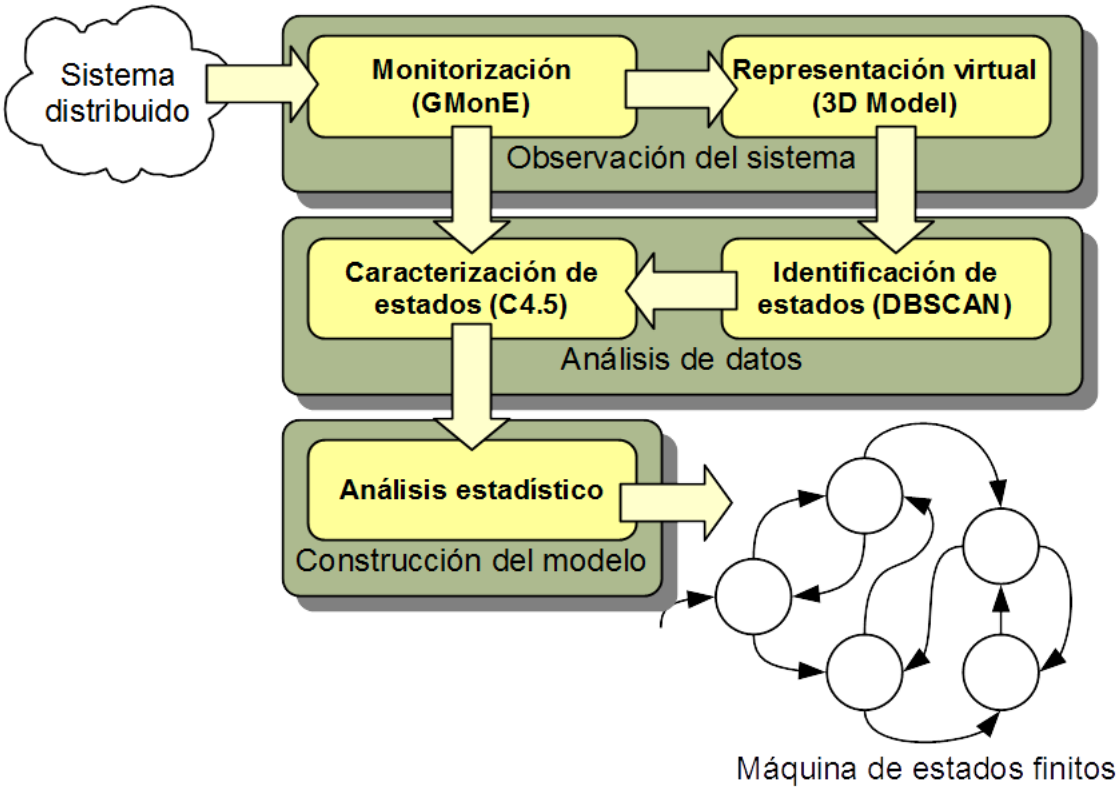


**Figura 1, Fases de GloBeM**

La Figura 1 muestra estas fases de forma detallada, incluyendo sus etapas internas. La descripción de cada una de ellas se realiza a continuación.

## 5.1 Fase 1, Observación del sistema

El primer paso para estudiar el comportamiento del sistema es observarlo. Mediante herramientas de monitorización, se recopila información en cada recurso que compone el sistema. El objetivo consiste en observar los distintos parámetros que pueden ser monitorizados (uso de CPU, uso de memoria, tráfico de red...) y agregarlos entonces para obtener medidas globales, mediante el uso de descriptores estadísticos como la media y la desviación típica.

A pesar de que esta información de monitorización es agregada en valores globales, normalmente la cantidad de datos obtenidos hace que se tenga que aplicar un mecanismo de representación de la información que permita manejar cómodamente estos datos y realizar su análisis.

Para esto, GloBeM hace uso de la *representación virtual de sistemas de información*, un método avanzado de extracción de conocimiento que proporciona un punto de vista distinto para el análisis de grandes volúmenes de datos. Se basa en crear una representación gráfica de un conjunto complejo de datos, reduciéndolos a 2 ó 3 dimensiones (fácilmente visualizables). Cada elemento de esta nueva representación corresponde a uno del conjunto de datos original, y su posición en el espacio resultante es relativa a su grado de similitud con el resto de elementos. Así pues, cuanto más parecidos sean dos elementos, más cerca aparecerán en la representación; y cuanto más distintos sean, más lejos. Esto a su vez, implica que aquellos instantes de monitorización que presenten valores similares aparecerán en la representación tridimensional formando densas *nubes*, separadas entre sí por zonas de menor concentración.

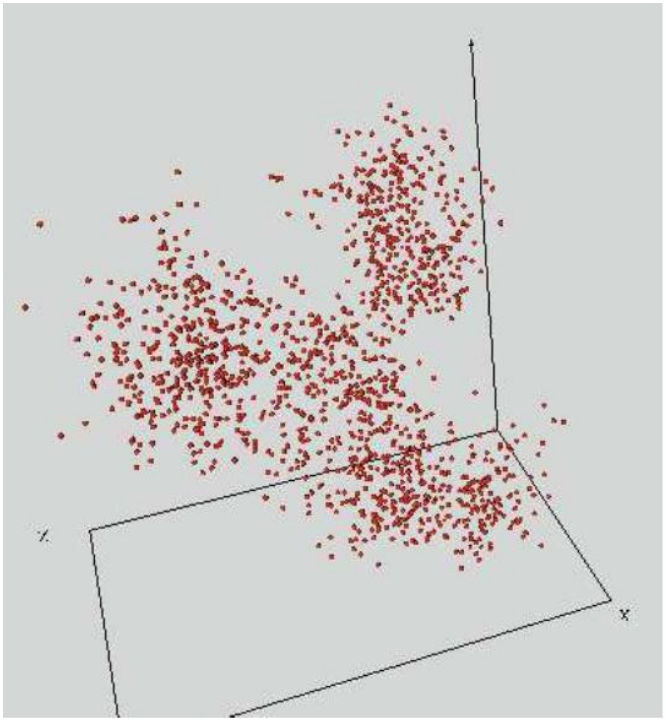

**Figura 2, Ejemplo de representación tridimensional**

La Figura 2 muestra un ejemplo del tipo de representación generada por esta técnica. Las *nubes* observadas indican posibles estados observados en el sistema y su análisis exhaustivo permitirá la construcción del modelo global de comportamiento.

## 5.2 Fase 2, Análisis de datos

En esta fase, se lleva a cabo la identificación y descripción de los estados globales del sistema a partir de la representación tridimensional obtenida en la fase previa.

Para identificar los distintos estados observados, es necesario *separar* las diferentes *nubes* de puntos encontradas en la representación visual. Para esto, GloBeM hace uso de herramientas de aprendizaje automático (relacionadas habitualmente con el campo de *data science*), capaces de extraer conocimiento no trivial de grandes conjuntos de datos. Concretamente, se sirve del algoritmo de *clustering* DBSCAN [7]. Se trata de un algoritmo especialmente diseñado para identificar variaciones de densidad en conjuntos de datos con un número bajo de dimensiones, por lo que encaja perfectamente con las necesidades del problema. La aplicación de este algoritmo proporciona automáticamente una separación de la representación tridimensional en grupos diferenciados, identificando los posibles *estados* del sistema.

A continuación, GloBeM hace uso del algoritmo C4.5 [8] aplicado a la información de los estados obtenida, más los datos originales, para llevar a cabo lo que, en terminología habitual de aprendizaje automático, se conoce como *clasificación supervisada*. Es decir, analizar los distintos grupos para entender su significado.

C4.5 es un clasificador estadístico que genera como resultado un árbol de decisión, que explica la clasificación indicada (los estados del sistema, en este caso). La interpretación de este árbol permite traducirlo a un conjunto de reglas de decisión que definan de forma inequívoca las condiciones que se deben cumplir para que el grid se encuentre en cada estado.

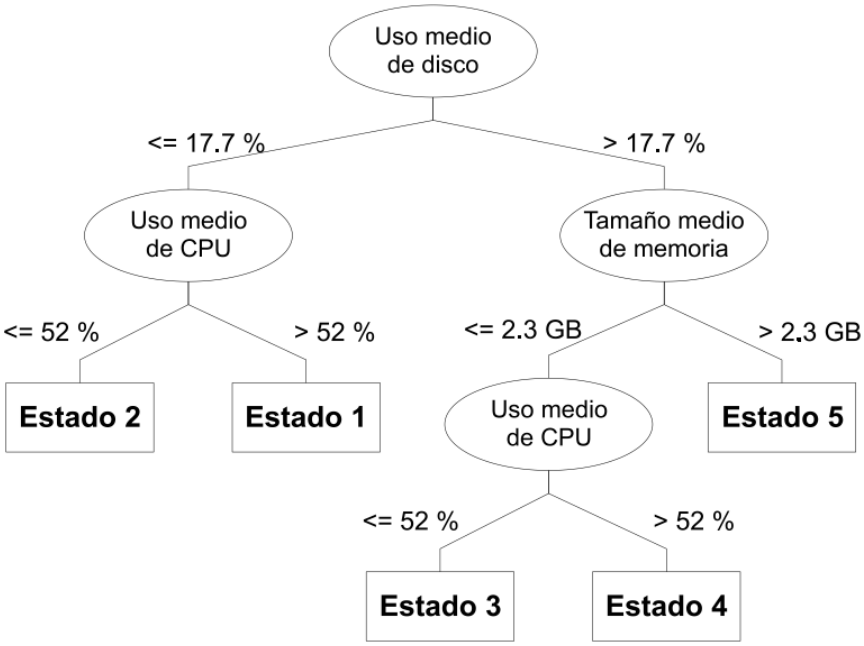


**Figura 3, Ejemplo de árbol de decisión obtenido con el algoritmo C4.5**

La figura 3 muestra un ejemplo de un posible árbol de decisión obtenido al aplicar el algoritmo C4.5.

## 5.3 Fase 3, Construcción del modelo

Finalmente, mediante una revisión del árbol de decisión anterior (el cual carece de ambigüedad) se pueden obtener las transiciones entre los estados, consiguiendo un modelo final que cumple con las características fundamentales de una máquina de estados finitos.

Además, cuenta con información estadística extra, que se puede adquirir al analizar estadísticamente los datos monitorizados, como la probabilidad de cada estado o las probabilidades de transición en estos.

Con la aplicación de todo este método sobre un entorno distribuido, se obtiene un modelo descriptivo del comportamiento de éste, considerándolo como un todo. Una vez conseguido, este modelo puede ser utilizado de diversas formas para llevar a cabo las tareas de gestión sobre el sistema estudiado. Y más concretamente, para poder tomar parte, por ejemplo, en el balanceo de carga.

Puesto que el sistema GloBeM sólo se encarga de la obtención del mencionado modelo, vamos a estudiar otro sistema más con la capacidad de encargarse del resto de la gestión del sistema.

# 6 Hydra

Hydra es un servicio de descubrimiento, gestión y balanceo de aplicaciones multi-cloud. Pretende facilitar el enrutamiento y el balanceo de carga de los servidores y delegarla en el cliente (navegador, aplicación móvil, etc.).

Hydra ha sido desarrollado por Innotech, bajo licencia de código abierto. El hecho de que sea una aplicación moderna y de código libre, perfectamente funcional en la gestión de entornos distribuidos, y con plena colaboración por parte de sus desarrolladores para resolver cualquier cuestión de la misma, ha hecho que sea seleccionada en este PFM para implementar la metodología GloBeM.

La herramienta se compone de diversos módulos, de los cuales, nos interesan principalmente los siguientes:

- **El servidor Hydra**. Éste es el núcleo del servicio. Se trata del módulo que lleva a cabo las principales tareas de descubrimiento de nubes y servidores, así como la gestión del balanceo de carga. Al menos, debe haber un servidor Hydra funcionando en un equipo y conectado con el resto de equipos del entorno (idealmente, un servidor por cada nube que tengamos, ya que los servidores Hydra pueden trabajar entre sí para gestionar varias nubes). Éste es también el módulo desde el que se controla la configuración del servicio y el modo de gestión del sistema, por lo que los módulos clientes se conectan con él para controlar las variables y opciones de gestión.
- **Las sondas**. Son pequeños servicios que se despliegan en aquellos servidores que queremos que formen parte de nuestra nube, los cuales (los servidores) tienen la capacidad de ejecutar las aplicaciones (todas o algunas) que son los servicios ofrecidos por el cloud hacia el exterior.
  Básicamente, lo que hacen estas sondas es darse de alta en el servidor Hydra de la nube, informando de la dirección del servidor en que están y la aplicación concreta que están monitorizando. Esto hace que necesitemos colocar una sonda por cada aplicación que deseemos monitorizar en cada servidor. A partir de ahí, la sonda se dedica a mandar información periódicamente al servidor de Hydra sobre el estado del equipo, es decir, el estado de la aplicación (si se sigue ejecutando y es accesible) y del propio equipo (los valores de carga de elementos como la CPU, la memoria, número de conexiones activas, y otros posibles parámetros definidos manualmente).
- **El balanceador**. Por último, el balanceador es la definición de un método para determinar el orden de preferencia de los servidores de la nube cuando llegan solicitudes de servicio a ésta. El balanceador es utilizado directamente por el servidor Hydra. Actualmente, los desarrolladores de esta herramienta han definido varios de estos balanceadores o *workers*, según su terminología propia, y un servidor Hydra puede usar sólo uno, o varios de ellos combinados, para obtener el balanceo de carga que determina qué equipos ejecutarán las peticiones de servicio cuando éstas ocurran. Algunos de los balanceadores creados actualmente para Hydra incluyen algoritmos sencillos como ordenar a partir del valor de un cierto atributo, o el algoritmo de round-robin.

La figura 4 muestra un esquema de la estructura de los módulos de Hydra y las relaciones de comunicación que se dan entre ellos. Se trata de un esquema muy general, donde podemos ver el uso de dos nubes, en cada una de las cuales hay una serie de

aplicaciones disponibles y un servidor Hydra que se encarga de gestionarlas. Los servidores Hydra de cada nube también se comunican entre ellos para sincronizar y coordinar su actividad. A su vez, se pueden ver los clientes Hydra, aplicaciones o bibliotecas que permiten el acceso y uso de las aplicaciones del sistema a través de los servidores Hydra (por ello, cada cliente está conectado a cada servidor Hydra). Los dos usuarios que aparecen usan las aplicaciones de las nubes de una forma transparente, sin importarles qué nube o equipo concreto está ejecutando dicha aplicación.

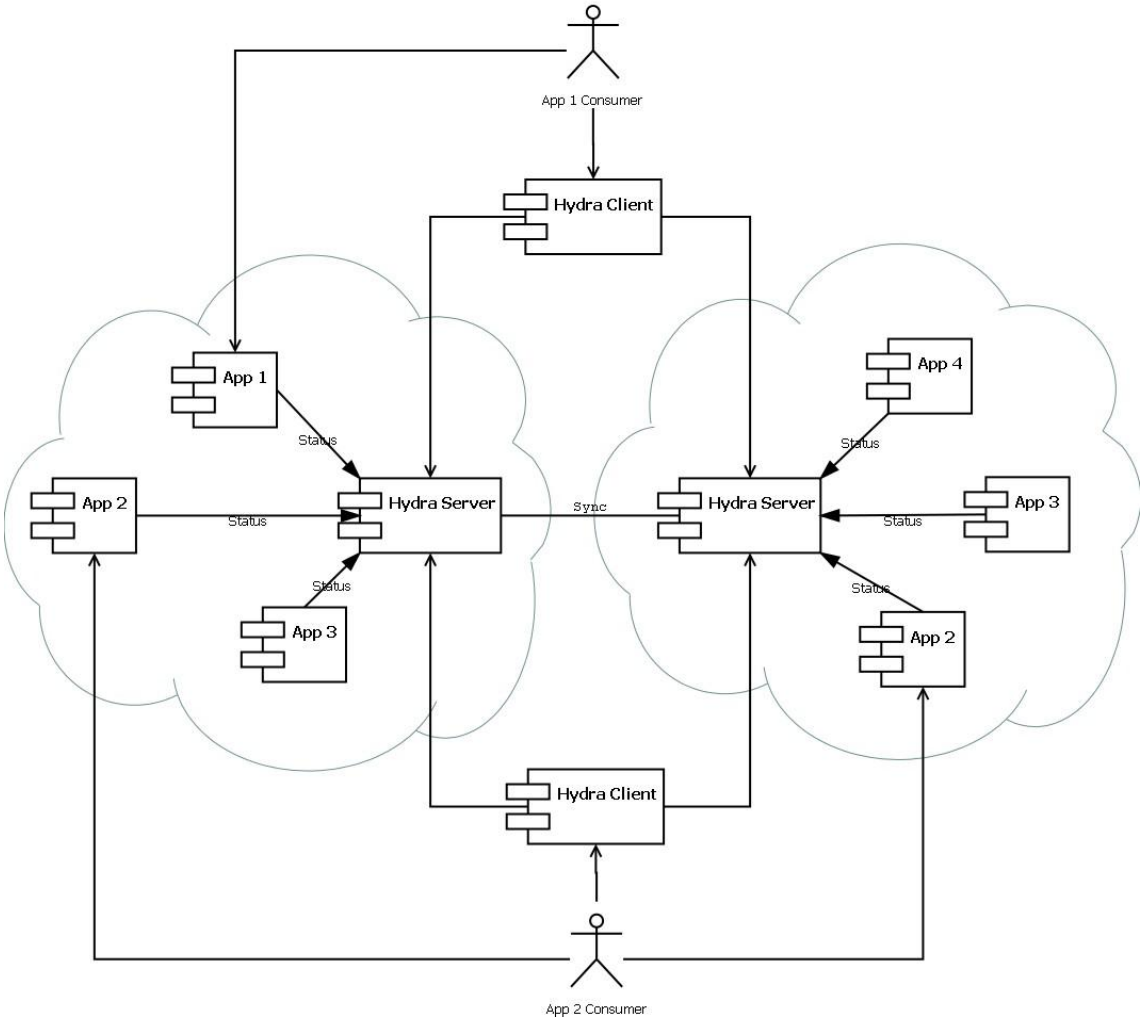


**Figura 4, Estructura de módulos de Hydra**

La figura 5 muestra un esquema de la arquitectura de Hydra:

- **Hydra Server**. El servidor de Hydra, formado por estos otros elementos o módulos:
    - *Embedded ETCD* [9], [10], [11], el sistema de pares clave-valor. Se explica a continuación.
    - *La base de datos de Hydra*.
    - *Load balancer*, el sistema de balanceo de carga que usa Hydra. Aquí se puede ver cómo se pueden asociar uno o más balanceadores o *workers* a este sistema de balanceo.

- *Server API y Client API*, que se corresponden con las interfaces para los módulos de las aplicaciones y los módulos de los clientes, respectivamente.

- **App Instances**. Se trata de las instancias de las aplicaciones que son ejecutables en diferentes equipos, las cuales son ofrecidas como servicios del sistema hacia el exterior. Para cada una de estas instancias, se coloca una sonda (*basic probe*) que es la encargada de mandar la información del estado del equipo al servidor Hydra.
- **Web Browser, Mobile Phone**… Éste es el entorno de ejecución del usuario final, el cual quiere hacer uso de las aplicaciones disponibles en el sistema. Para ello, hace uso del ya mencionado cliente Hydra (*Hydra Client*).

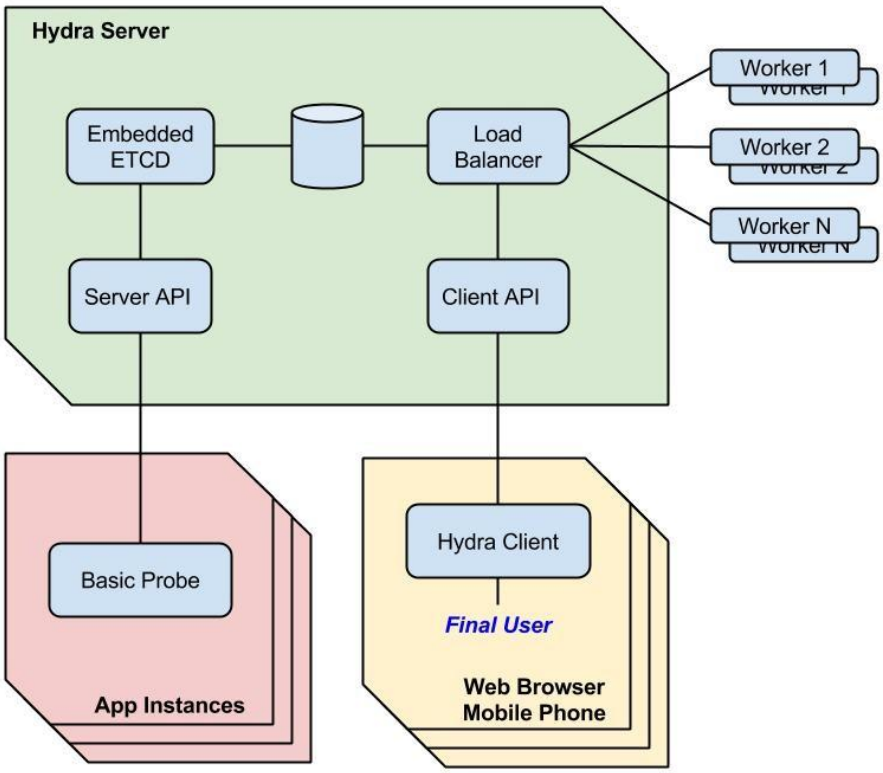


Figura 5, Diseño arquitectónico de Hydra

Actualmente, el sistema Hydra está construido para poder ser ejecutado únicamente en sistemas Linux. La comunicación entre sus diversos módulos y con el lado del cliente, se realiza a través de solicitudes web con el uso del protocolo HTTP. Estas solicitudes HTTP usan el formato marcado por *etcd*.

## 6.1 etcd

ectd es un almacén de pares clave-valor, distribuido como código abierto, que puede servir como la columna vertebral de un sistema distribuido, al proporcionar un centro estándar para la coordinación del clúster y su gestión. Aunque fue construido

específicamente para clústeres ejecutando CoreOS, actualmente funciona sobre una variedad de sistemas operativos, incluyendo OS X, Linux y BSD.

Los clústeres generalmente se construyen a partir de una gran colección de máquinas con la capacidad de ejecutar cualquier carga de trabajo en un momento dado. Y para que un clúster pueda funcionar de manera eficiente, necesita crear un sistema de comunicación y coordinación entre sus nodos, de forma que sea posible conocer y gestionar el estado del clúster en cada momento. Aquí es donde entra etcd.

La gestión del estado del clúster a través de cualquier sistema distribuido está plagada de numerosas situaciones que se deben tener en cuenta, muchas de las cuales tienen el potencial de crear problemas. Por ejemplo, los clústeres a menudo sufren de particiones de red y condiciones de carrera, los cuales necesitan ser resueltos en alguna parte. Gestionar esto desde las aplicaciones o a nivel de cada máquina individual resultaría demasiado duro, así que es a un nivel más alto donde etcd trabaja.

Para esto, etcd utiliza un sistema de almacenamiento de pares clave-valor como si de una base de datos se tratara. La consulta y modificación de los datos ahí contenidos se realiza a través de solicitudes HTTP, en cuya URL se utiliza un formato característico para la acción que se quiera realizar. Este sistema es usado tanto por Hydra internamente a la hora de realizar la gestión del clúster, así como externamente por los usuarios que quieran consultar información sobre el estado del mismo o cambiar la configuración de Hydra (como por ejemplo, modificar en un momento dado los balanceadores que Hydra está usando para el balanceo de carga).

Con todo lo dicho, tenemos una visión general de lo que son los entornos distribuidos, y algunas de las herramientas que actualmente sirven para su gestión y uso. Se ha explicado también el sistema GloBeM para el análisis de entornos distribuidos, y el servicio Hydra para la gestión de estos. Ahora podemos pasar a describir cómo se ha hecho frente al objetivo primordial de este trabajo: conseguir que una herramienta de gestión de entornos distribuidos como Hydra, sea capaz de llevar a cabo el balanceo de carga de dichos entornos basándose en un modelo de comportamiento que el método GloBeM es capaz de generar.

*Parte II*

# Estudio y solución del problema

*Especificación de requisitos*

*Descripción del diseño*

*Plan de pruebas*

# 7 Especificación de requisitos

El propósito de esta sección es describir la extracción de los requisitos para la aplicación objetivo de este TFM, con los que posteriormente se realizó el diseño e implementación de la misma.

Esta especificación de requisitos sigue, a grandes rasgos, las prácticas recomendadas descritas en el estándar '*IEEE Std. 830-1998, IEEE Recommended Practice for Software Requirements Specifications*' (Prácticas Recomendadas para la Especificación de Requisitos Software) [12]. Aunque se sigue el esquema general de éste, se ha simplificado su estructura para no crear un documento excesivamente técnico y extenso, conservando los elementos fundamentales.

De este modo, esta sección contiene dos subsecciones principales: la primera, que da una visión general del software y explica brevemente algunas de sus características; y la segunda, que entra más en detalle a exponer los requisitos y las funciones que los satisfarán.

Nótese que, como sucede habitualmente durante en el proceso de especificación de requisitos, no todos los requisitos aquí expresados se pudieron determinar en el momento del análisis para su extracción. Algunos de ellos se fijaron, o se expresaron con el suficiente detalle, una vez se había podido avanzar en cierta medida en el diseño del software.

## 7.1 Descripción general

### 7.1.1 Perspectiva de la herramienta

El objetivo de este proyecto es el de implementar una herramienta capaz de monitorizar y gestionar la actividad del servicio Hydra, de modo que la gestión del balanceo de carga que ésta realice se adecúe a un modelo definido por el sistema GloBeM. Por lo tanto, tanto la modelización del sistema cloud gestionado por Hydra, incluyendo la determinación de los estados y el diseño del mencionado modelo se considera un paso previo, que se debe realizar de manera externa al sistema desarrollado en este proyecto. Para ser utilizado por parte de la herramienta, este modelo estará definido como un conjunto de políticas, las cuales determinarán en cada momento la estrategia de balanceo que se debe seguir, en función del estado del sistema.

Esto implica que la herramienta debe disponer de varias interfaces para comunicarse con diversos elementos. El primero de ellos es el resultado final de la aplicación de GloBeM, lo cual se convierte en la entrada a la herramienta que nos concierne, que es la definición de las políticas de balanceo. La definición de estas políticas se realiza para cada aplicación que se ejecute en el cloud, y se basa en determinar una serie de estados en los que se pueden encontrar los nodos del sistema que gestiona Hydra, capaces de ejecutar la aplicación en cuestión. Además, se definen unos balanceadores que se aplicarán en la configuración de Hydra para que sirvan como estrategia de balanceo de carga. Según en qué estado se encuentre el sistema en cada momento, se reconfigurará Hydra para aplicar unos balanceadores u otros.

El segundo elemento con el que deberá poder comunicarse es con el usuario. Se va a tratar de una herramienta que deberá llevar a cabo sus funciones de forma autónoma, de manera que no requiera de una supervisión en tiempo real y continua de su ejecución. Por lo tanto, lo que se requiere de la herramienta es que pueda informar al usuario ante los diferentes eventos que puedan suceder, de forma que éstos se puedan analizar en el momento en que se requiera por el usuario o administrador del sistema. Esta información incluye principalmente errores y alertas de diversa índole, acontecidos en tiempo de ejecución.

Por otro lado, la herramienta también debe disponer una interfaz que permita al usuario modificar ciertas características de la misma, sin necesidad de tener que modificar el código, recompilar, etc. Esto se refiere a opciones de configuración como son fijar la frecuencia con la que se debe monitorizar el sistema Hydra, especificar dirección de red del servidor en donde se ejecuta éste, etc.

Por último, la herramienta deberá poder comunicarse con el ya mencionado sistema Hydra, para así poder gestionar su configuración y comportamiento.

### 7.1.2 Funciones

Como ya se ha mencionado, la función principal de la herramienta será la de monitorizar y gestionar el comportamiento del sistema Hydra, basándose en las políticas de balanceo creadas.

Para ello, el software deberá:

- Obtener e incorporar a su sistema de datos toda la información referente a las políticas de balanceo.
- Comunicarse con Hydra y obtener, en tiempo real, la información de monitorización de los nodos que componen el cloud manejado por Hydra.
- Para cada aplicación en ejecución, determinar en qué estado se encuentran los nodos del cloud que la ejecutan según la políticas definidas.
- A continuación, reconfigurar Hydra para aplicar las políticas de balanceo para cada aplicación, según el estado en que se haya determinado que se encuentran.

### 7.1.3 Usuarios

Esta herramienta está destinada a administradores de sistemas cloud. Concretamente, aquellos que, además, poseen conocimientos sobre el sistema Hydra y su funcionamiento, ya que se trata de una herramienta que complementa, mejora y amplía sus prestaciones.

Por lo tanto, es de esperar que el usuario de esta herramienta posea amplios conocimientos sobre entornos cloud, así como sobre comunicaciones de red, y por supuesto, sobre Hydra.

### 7.1.4 Restricciones y limitaciones

Como ya se ha mencionado, la monitorización se deberá llevar a cabo en tiempo real. Esto implica que la herramienta tendrá que solicitar los datos a Hydra con una frecuencia relativamente alta, y por lo tanto, el rendimiento a la hora de efectuar las operaciones es de crucial importancia para poder respetar dicha frecuencia.

Por otro lado, tanto las características configurables de la herramienta, como las políticas de balanceo, deben de poderse cambiar en cualquier momento, sin que esto implique tener que modificar el código, recompilar o ni siquiera detener la ejecución de la herramienta en ningún momento.

Por último, el software deberá ser capaz de alertar de cualquier problema percibido con cualquiera de las entradas al sistema lo más pronto posible, de modo que el usuario pueda encontrar fácilmente dicho error y solucionarlo. Esto implica que todas las entradas al sistema deben evaluarse correctamente antes de añadirse a los sistemas de información internos, para así informar de posibles errores de manera inmediata, y de forma tal que sea comprensible y solucionable por parte del usuario del sistema.

Además de esto, el software deberá ser tolerante a los fallos en la media de lo posible. Que la herramienta siga funcionando en todo momento a pesar de los fallos que puedan surgir es crítico para que resulte una herramienta confiable. Por lo tanto, deberá ser capaz de recoger y mostrar cualquier excepción, fallo o error que perciba durante la ejecución, y a continuación continuar su desempeño desde el último punto en que éste funcionó correctamente. La decisión de finalizar la ejecución del software debe recaer en manos del administrador.

### 7.1.5 Suposiciones y dependencias

La principal suposición o dependencia que se va a hacer es en cuanto al sistema operativo que ejecutará la aplicación. Puesto que Hydra está diseñado para el sistema operativo Linux, y ya que Windows es el más extendido de todos, para la creación de esta herramienta sólo se van a tener en cuenta estos dos sistemas operativos, y es en los únicos en que se espera que pueda funcionar correctamente.

## 7.2 Requisitos específicos

### 7.2.1 Interfaces externas

Como ya se ha mencionado, la herramienta va a contar con cuatro interfaces de entrada-salida para comunicarse con diferentes elementos con los que debe trabajar o a los que debe informar:

- Interfaz de entrada para la información de las políticas de balanceo.
- Interfaz de entrada para la configuración de la herramienta por parte del usuario.
- Interfaz de salida para informar al usuario de los sucesos que acontecen durante la ejecución.
- Interfaz de entrada-salida con Hydra para las operaciones de monitorización y reconfiguración del citado sistema cloud.

#### 7.2.1.1 Definición de las políticas

La información sobre las políticas de balanceo se obtiene como salida del sistema GloBeM. Se ha decidido que, puesto que se trata de un sistema independiente, la comunicación se realice a través de un fichero que contenga la información sobre las políticas, cuyo formato a utilizar sea JSON [13]. Esta decisión se basa en la simplicidad de éste, la gran cantidad de bibliotecas existentes que permiten trabajar con él, y porque el sistema Hydra también lo utiliza.

En este fichero se deben describir principalmente dos cosas: los estados de las aplicaciones (incluyendo las condiciones que hacen que se cumplan unos estados u otros); y la descripción de los balanceadores que se deben de usar cuando se cumplan dichos estados.

Por tanto, el fichero contendrá dos grandes secciones (‘balanceadores’ y ‘aplicaciones’) y su estructura, en JSON, será la siguiente:

```
[{"Balancers": [
      {"name": "balancerName",
      "parameters": [
            {"name": "parameterName",
            "value": "parameterValue"
            },
            ...
      ]},
      ...
]},
{"Apps": [
      {"name": "aplicationName",
      "states": [
            {"name": "stateName",
            "condition": "condition",
            "balancers": "listOfBalancersName"
            },
            ...
      ]},
      ...
]}]
```

JSON posee una estructura de pares clave-valor. En la estructura mostrada, las claves representan las palabras reservadas que identifican los diferentes campos de información que componen las políticas, mientras que los valores serán sustituidos por los valores reales de esos campos.

Como se puede ver, el estado se determina por cada aplicación que se ejecuta en el cloud sobre el conjunto de nodos capaces de ejecutar dicha aplicación. Un punto muy importante que se debe exponer aquí es el de las condiciones de dichos estados. Para determinar en qué estado se encuentran los nodos para una aplicación concreta se evalúa su condición, que es una expresión lógica/matemática que utiliza como variables el resultado de aplicar unas operaciones algebraicas a los valores en tiempo real de las

variables de monitorización que maneja Hydra para el conjunto de nodos que ejecutan la mencionada aplicación.

Estas operaciones pueden ser de cuatro tipos:

- Media aritmética
- Desviación típica
- Máximo
- Mínimo

Y las variables de monitorización sobre las que se aplican son:

- Carga de la CPU
- Carga de la memoria
- Número de conexiones de red

Como existe libertad total para decidir el formato que tendrán las expresiones matemáticas que se deben evaluar, se ha decidido que se usará el 'estándar' que define la biblioteca JEval [14], [15], la cual permite evaluar expresiones lógicas y algebraicas de manera sencilla, y que además, cubre todo el rango de operaciones que son necesarias para las condiciones que aquí se van a utilizar.

El anexo A proporciona una descripción del formato que deben seguir las expresiones matemáticas de las condiciones para poder ser evaluadas mediante la biblioteca JEval.

El anexo B muestra un ejemplo sencillo del contenido del archivo de políticas en su formato JSON.

#### *7.2.1.2 Configuración de la herramienta*

La configuración de la herramienta se va a realizar a través de un archivo que contendrá las diferentes opciones de configuración editables junto con sus valores. Tendrá un formato simple y de fácil creación por parte de un usuario. En este caso, no se usa el formato JSON, ya que, aunque éste es muy útil para ser creado y leído de forma procedural mediante el uso de bibliotecas, resulta mucho más incómodo si debe ser creado manualmente. Por lo tanto, el archivo de configuración será un archivo de texto, compuesto por el listado de pares claves-valor, en el que habrá un par por cada línea, de la siguiente forma:

```
option1=value
option2=value
...
```

Este sencillo formato hace que resulte fácil e intuitivo tanto escribir las opciones de configuración, como leerlas.

Tras avanzar en cierta medida en el diseño, se pudo determinar que las opciones que son necesarias que aparezcan en este fichero son:

- El nombre y ruta del archivo con la definición de las políticas.
- El intervalo de tiempo que marca la frecuencia con la que se debe consultar la información del sistema Hydra, y reconfigurarlo en su caso.
- La dirección de red del servidor donde se está ejecutando Hydra.
- El tiempo de espera para intentar realizar la comunicación con Hydra, antes de desistir y entender que se ha podido producir un fallo.

#### 7.2.1.3 Información de sucesos

Puesto que uno de los objetivos de este trabajo es que la herramienta desarrollada pueda trabajar de manera autónoma sin necesidad de un usuario que la supervise continuamente, para la notificación de los eventos que tienen lugar durante la ejecución se ha decidido hacer uso de un fichero de registro. De esta forma, toda la información relevante se mostrará a través de la consola de mandatos, así como en un fichero en memoria no volátil (disco) donde queden registrados todos estos datos para posteriores revisiones.

El formato de los eventos registrados en el fichero incluirá la fecha y hora exacta del evento, un nivel de gravedad asignado, y la descripción del error que permita al usuario ponerle solución. No se incluirán trazas de líneas de código ni mensajes genéricos que puedan confundir al usuario o que no le sirvan para corregir la situación que se ha presentado.

Por otro lado, el nombre del archivo vendrá determinado por el día en que se crearon sus registros. El formato será el siguiente:

AAAA-MM-DD-log

Donde:

- AAAA: año en formato numérico
- MM: mes en formato numérico
- DD: día en formato numérico

En cuanto a qué mensajes se mostrarán aquí concretamente, se describe en la siguiente sección ‘Funciones’.

#### 7.2.1.4 Comunicación con Hydra

Las comunicaciones que se realizarán con Hydra van orientadas en dos sentidos: obtener todos los valores de las variables de monitorización en el momento en que se soliciten y reconfigurar Hydra para que ésta incluya los nuevos balanceadores descritos por las políticas de balanceo.

La interfaz de comunicación con el sistema Hydra viene determinada por su propia especificación, la cual ya se mencionó que usa el sistema etcd (HTTP + JSON). Por lo tanto, la herramienta deberá ser capaz de establecer comunicaciones de red con el servidor donde se ejecute Hydra (que en ocasiones podrá ser el mismo equipo, o puede

tratarse de equipos diferentes) utilizando el protocolo de comunicaciones HTTP y el formato definido por etcd para la creación de las URLs HTTP.

Los datos devueltos por Hydra siempre tienen un formato JSON.

El anexo C muestra un ejemplo sencillo de los datos devueltos por Hydra, en formato JSON, al solicitar la información del sistema cloud gestionado.

### 7.2.2 Funciones

Esta sección contiene una descripción más detalla de las funciones que debe realizar la herramienta.

#### *7.2.2.1 Monitorizar y gestionar Hydra*

Se trata de la característica o función principal de la herramienta. Cada cierto tiempo, el software debe iniciar por sí mismo su actividad, y llevar a cabo las siguientes tareas:

1. Comunicarse con Hydra para obtener toda la información de las variables de monitorización de los nodos que gestiona.
2. Analizar y formatear dicha información para introducirla en estructuras de datos que permitan un mejor acceso a ella.
3. Analizar una por una las aplicaciones de las que Hydra ha devuelto datos.
4. Para cada una, debe obtener los valores de las variables de monitorización de los nodos o instancias asociados a cada aplicación.
5. Sobre esos conjuntos de valores, aplicar las operaciones estadísticas para la evaluación de las condiciones expresadas en las políticas.
6. Examinar uno por uno los estados de las aplicaciones, evaluando para ello sus condiciones con los valores antes obtenidos, hasta obtener el estado que se haga cierto.
7. Comunicarse con Hydra para eliminar los balanceadores ya obsoletos asociados a la aplicación, si los hay.
8. Comunicarse con Hydra para definir los nuevos balanceadores descritos en las políticas, en caso de que sea necesario.

Implicaciones:

- La actividad de la herramienta vendrá determinada por ciclos de ejecución. Durante estos, la herramienta llevará a cabo todas las funciones que sean precisas; y al no poder realizar más actividades dentro de un mismo ciclo, la ejecución pasará a estar bloqueada durante un tiempo determinado (parámetro configurable ‘frecuencia de ejecución’) hasta poder reanudar de nuevo la ejecución en un nuevo ciclo.

#### *7.2.2.2 Informar de anomalías durante la comunicación con Hydra*

Siempre que un sistema informático cuenta con alguna sección en el mismo que implique el uso de comunicaciones de red, se debe tener en cuenta la posibilidad de que dicha comunicación falle y no se puedan recibir o enviar los datos que se esperaba.

En este caso, los motivos por los que puede fallar la comunicación son muy diversos. Lo importante, no obstante, es identificar dónde se producen esos fallos. En este sentido, existen dos momentos críticos en el sistema:

- Cuando se solicitan los datos de monitorización de Hydra
- Cuando se reconfiguran los balanceadores de Hydra (eliminar antiguos, añadir nuevos).

Cualquier error que se produzca en uno de estos puntos se debe registrar, añadiendo como información adicional la ofrecida por la excepción recogida durante el intento de comunicación.

Implicaciones:

- Si el fallo se produce al solicitar los datos de monitorización, el ciclo entero de ejecución se debe terminar, puesto que no se dispondrá de los datos de Hydra.
- Si el fallo se produce durante la reconfiguración de los balanceadores de Hydra, se debe terminar la configuración únicamente de la aplicación que se estaba tratando en ese momento (recordemos que las aplicaciones se reconfiguran en Hydra una por una y por separado). Así, puede que haya fallado la comunicación para una, pero no para todas.

#### *7.2.2.3 Obtener los datos de configuración*

La herramienta debe ser configurable en ciertos aspectos, por lo cual, ciertas variables de su comportamiento deben poder modificarse y trabajar con nuevos valores en cualquier momento. Para ello, la herramienta deberá realizar con regularidad lo siguiente:

1. Leer el archivo de configuración.
2. Añadir los valores nuevos a los parámetros de la herramienta para trabajar con ellos.

Implicaciones:

- Para que los datos de configuración sean actualizables en cualquier momento, el sistema debe ser capaz de detectar cambios en el fichero de configuración.

#### *7.2.2.4 Informar de situaciones anormales en el archivo de configuración*

La información del archivo de configuración debe ser verificada y validada antes de añadirse al sistema. Cualquier error haría que la información contenida en él no se añadiera al sistema, y que, además, se informara del error exacto encontrado.

Las situaciones anómalas que pueden causar un error durante la lectura del archivo de configuración, y que por lo tanto deben ser tenidas en cuenta, son las siguientes:

- No existe el archivo.
- El archivo está vacío.

- Contiene las variables correctas que posteriormente serán utilizadas, junto con valores incorrectos.
- Contiene variables incorrectas, es decir, que posteriormente no serán utilizadas durante la ejecución del programa.
- La definición de alguna variable no es correcta:
    - no se usa el separador '=';
    - hay más de una definición en una misma línea;
    - hay variables repetidas.

Implicaciones:

- Los datos de configuración de la herramienta, en un momento dado, se podrán sustituir por otros datos nuevos leídos si, y sólo si, la lectura del archivo fue correcta en su totalidad.
- Si se produce algún error durante la lectura inicial del archivo, los parámetros configurables de la herramienta quedarían sin ningún valor, y no habría posibilidad alguna de ejecución. Para evitar esto, la herramienta contará inicialmente con unos valores de configuración por defecto.

#### *7.2.2.5 Obtener los datos de las políticas*

Para poder llevar a cabo la actividad principal de la herramienta, *Monitorizar y gestionar Hydra*, es necesario que se hayan leído los datos de las políticas, que será mediante los que se aplique la nueva configuración. Para ello, el software debe:

1. Leer el archivo de políticas de balanceo.
2. Analizar y formatear su información para introducirla en estructuras de datos que permitan un mejor acceso a ella.

Implicaciones:

- Para poder llevar a cabo las actividades del requisito '1. Monitorizar y gestionar Hydra', es necesario que primero se haya podido ejecutar con éxito esta función.
- Para que los datos de las políticas sean actualizables en cualquier momento, estos deben ser leídos en cada ciclo de ejecución para asegurar que se posee la última versión de los mismos.

#### *7.2.2.6 Informar de situaciones anómalas en el archivo de políticas*

Al igual que con el de configuración, toda la información contenida en el archivo de políticas de ser verificada y validada antes de incorporarse al sistema para su posterior uso. Cualquier error haría que la información contenida en él no se añadiera al sistema, y que, además, se informara del error exacto encontrado.

Las situaciones anómalas que pueden causar un error durante la lectura del archivo de políticas, y que por lo tanto deben ser tenidas en cuenta, son las siguientes:

- No existe el archivo.
- El archivo está vacío.

- Falta la definición explícita de una o ambas secciones del archivo (balanceadores y aplicaciones).
- La estructura del archivo, en forma de árbol, no es la esperada:
    - Alguna palabra reservada está mal escrita.
    - Algún elemento no aparece donde debería.
    - Hay algún elemento de más de lo que se espera recibir.
- En el archivo se definen más aplicaciones que las que está manejando Hydra.
- En el archivo se definen menos aplicaciones que las que está manejando Hydra.
- En el archivo no se definen todos los balanceadores que son referenciados en la sección de aplicaciones.
- En el archivo se definen más balanceadores de los que son referenciados en la sección de aplicaciones.
- La definición de la expresión lógica/matemática de alguna de las condiciones de los estados de las aplicaciones es incorrecta, según el formato que determina la biblioteca *JEval*.
- En las condiciones, aparece alguna variable cuya operación no se reconoce entre el conjunto de posibles operaciones que realizar sobre las variables de monitorización.
- En las condiciones, aparece alguna variable cuya variable de monitorización asociada no aparece entre las que ofrece Hydra sobre las aplicaciones.
- El tipo de datos de una variable de monitorización que aparece en una variable no coincide con lo que espera recibir la operación a realizar sobre ella.
- Existe una aplicación para la cual no se cumple ninguna de las condiciones de sus estados.
- Existe una aplicación para la cual se cumplen dos o más de las condiciones de sus estados.

Implicaciones:

- Los datos de las políticas de balanceo, en un momento dado, se podrán sustituir por otros datos nuevo leídos si, y sólo si, la lectura del archivo fue correcta en su totalidad.
- Si se produce algún error durante la lectura inicial del archivo, la herramienta no contaría con datos de políticas con los que gestionar Hydra.

### 7.2.3 Requisitos de rendimiento

Como ya se ha mencionado, esta herramienta debe permanecer constantemente en ejecución, monitorizando en tiempo real la situación de Hydra, lo cual implica que debe llevar a cabo sus tareas de forma periódica y con una frecuencia relativamente alta.

Concretamente, se espera que la frecuencia de su actividad varíe entre los 30 segundos hasta los pocos minutos (recordemos que éste es un parámetro configurable al gusto del administrador/usuario).

Por esta razón, es preciso minimizar los tiempos de procesado de los ciclos de ejecución en que se divide la actividad de la herramienta. En este sentido, la norma de referencia a tener en cuenta será la siguiente:

*El 95% de los ciclos de ejecución deben finalizar en menos de un minuto.*

Este va a ser el único requisito de rendimiento para la herramienta. Para cumplirlo, se deberán cumplir, al menos, las siguientes condiciones:

1. En cada ciclo, la lectura del archivo de configuración sólo se realizará en el caso de que éste haya sido modificado. De este modo, se pueden ahorrar muchos accesos a memoria secundaria.
2. En cada ciclo, la lectura del archivo de políticas sólo se realizará en el caso de que éste haya sido modificado, o si se ha especificado un nuevo archivo en los parámetros de configuración. De este modo, se pueden ahorrar muchos accesos a memoria secundaria.
3. Puesto que la frecuencia de monitorización va a ser relativamente alta, no se espera que los estados determinados para cada aplicación en cada ciclo de ejecución cambien constantemente. Por lo tanto, se deberá guardar, para cada aplicación, el estado que se obtuvo en el ciclo de ejecución precedente, y compararlo con el estado que se ha determinado en el ciclo actual. Si resulta ser el mismo (es decir, el estado en que se encuentran los nodos del cloud, según lo establecido en las políticas de balanceo, para la aplicación en cuestión no ha cambiado), entonces no sería necesario reconfigurar los balanceadores de Hydra. De este modo, se reducen las comunicaciones de red.
4. En cuanto a las condiciones de los estados, éstas pueden utilizar en su definición sólo un subconjunto de las operaciones estadísticas disponibles sobre un subconjunto del total de variables de monitorización que ofrece Hydra. Por ello, los cálculos de dichas operaciones sobre dichas variables no se debe realizar al principio, al recibir los datos procedentes de Hydra, sino que se deben efectuar en el momento en que se precisen. De este modo, se pueden ahorrar cálculos de las operaciones estadísticas sobre los valores de las variables de monitorización que luego no serían utilizados para la evaluación de las condiciones de los estados.

Por lo demás, no se van a establecer más requisitos específicos de rendimiento, más allá de los que se le exige a cualquier buen producto de ingeniería, donde siempre se debe buscar un uso eficiente y eficaz de los recursos.

### 7.2.4 Limitaciones de diseño

Para la creación de esta herramienta, la limitación más importante que se debe tener en cuenta tiene que ver con el protocolo de comunicación con Hydra. Para éste, Hydra hace uso de etcd, una herramienta de almacenamiento de pares clave-valor, que permite el intercambio de dichos pares mediante el paso de documentos JSON a través de solicitudes HTTP. El problema es que dichas solicitudes son extremadamente simples, y

no permiten el uso de peticiones más complejas como las que permitiría, por ejemplo, un sistema gestor de bases de datos con interfaz SQL.

Esto tiene dos implicaciones:

1. La definición de los balanceadores y sus parámetros en Hydra se debe hacer en solicitudes separadas. Es decir, una solicitud HTTP para definir el balanceador, y otra por cada parámetro y su valor correspondiente. Esto hace que la definición de un solo balanceador suponga varias peticiones HTTP, en vez de poder hacerlo en una sola. Por lo tanto, esta situación refuerza, aún más si cabe, la necesidad de guardar el estado del ciclo de ejecución precedente, como se expuso en el tercer requisito de rendimiento del apartado anterior.
2. Estas peticiones HTTP a la herramienta etcd no permiten el establecimiento de una transacción, es decir, realizar una serie de peticiones relacionadas unas con otras con la característica de que si una sola de ellas falla, todos los cambios de la transacción se revierten y la base de datos (de Hydra) quedan sin modificar. Puesto que las transacciones, por su naturaleza, deben ser ejecutadas en el entorno del servidor (es decir, no hay forma de que sean programadas y ejecutadas por el cliente que hace las solicitudes), si se produce un fallo de comunicación durante el envío de alguno de los mensajes que determinan la configuración completa de un balanceador en Hydra, dicho balanceador quedará incompleto en su base de datos y puede provocar el mal funcionamiento del sistema.
   Este problema puede intentar atajarse al establecer una frecuencia de monitorización alta, lo cual permita que si se ha producido el fallo descrito, éste se repare rápidamente en el siguiente ciclo de ejecución, y así reducir en la medida de lo posible el tiempo en que Hydra se estará ejecutando con una base de datos de balanceadores incorrecta.
   Por otro lado, y como ya se ha mencionado, este fallo debe ser registrado para que quede constancia de él.

La manera más efectiva para evitar que se produzca la desagradable segunda situación descrita, es que esta herramienta de monitorización y gestión de Hydra se ejecute siempre en el mismo equipo que el propio sistema Hydra, de modo que los fallos de comunicación quedarían reducidos al mínimo.

### 7.2.5 Atributos del sistema

Para finalizar, en este apartado se va a hablar de los atributos software para este sistema. Estos atributos incluyen la fiabilidad, disponibilidad, usabilidad, mantenibilidad, seguridad, etc.

De todos los posibles, sólo se precisan dos: mantenibilidad y portabilidad.

#### *7.2.5.1 Mantenibilidad*

El sistema Hydra de gestión de entornos cloud se distribuye actualmente bajo licencia de código abierto, de modo que puede ser usado y modificado por cualquier persona.

Se ha decidido que esta herramienta va a seguir la misma filosofía, y se tratará de un proyecto de código abierto, que igualmente, podrá ser usado y modificado por cualquier persona. Así, el atributo de mantenibilidad es crítico para que esto se pueda cumplir.

Para ello, se prescriben las siguientes pautas:

- Elegir un paradigma de programación que haga especial hincapié en la organización y estructuración comprensiva del código, como es, por ejemplo, el paradigma de orientación a objetos.
- Crear un código los más limpio y comprensible posible, evitando las líneas de código excesivamente largas y complejas.
- Añadir comentarios y explicaciones a lo largo de todo el código.
- Estas anotaciones, junto con los nombres utilizados para identificar clases, métodos, atributos, etc., se deberán escribir utilizando como idioma informal el inglés (de modo que su uso y distribución resulten más 'internacionales'). Quedan aquí incluidos los mensajes mostrados en las interfaces externas descritas con anterioridad.
- Los módulos que implementen la funcionalidad de las operaciones estadísticas ya descritas, que se realizan sobre las variables de monitorización de Hydra, se deberán construir de manera que resulte sencillo añadir nuevas operaciones, para su uso en la aplicación, con el mínimo esfuerzo posible.

#### *7.2.5.2 Portabilidad*

Como ya se mencionó anteriormente, se requiere que la herramienta pueda ejecutarse, al menos en principio, en dos sistemas operativos: Windows y Linux. Pero no se desea tener que crear el programa dos veces para adaptarlo a cada uno de los dos sistemas.

Esto implica, por un lado, que se limita el uso de los lenguajes de programación a alguno que sea portable entre sistemas operativos (como es el caso de Java, por ejemplo).

Y por otro lado, que durante la construcción del código, se deberá tener en cuenta que no se puede asumir que se está en un sistema operativo concreto, si no que se trata de un código que debe ser válido, al menos, para ejecutarse correctamente en los dos sistemas operativos citados.

# 8 Descripción del diseño

El propósito de esta sección es describir el diseño software sobre el cual se ha construido la herramienta controladora del sistema Hydra.

Esta descripción sigue, a grandes rasgos, las prácticas recomendadas descritas en el estándar '*IEEE Std. 1016-2009, IEEE Standard for Information Technology – Systems Design – Software Design Descriptions*' (Estándar para la Descripción de Diseños Software) [16]. Aunque se sigue el esquema general de éste, se han abreviado o simplificado algunas secciones para no crear un documento excesivamente extenso. Es por ello que sólo se han incluido algunas de las vistas más descriptivas del diseño, haciendo más hincapié en mostrar su estructura y menos en su lógica de trabajo (es decir, lo que hace, puesto que esto ya se vio en la sección anterior de requisitos donde se definían las funciones que se le requerían al software).

Así, primero se describirán las herramientas y lenguajes de desarrollo que se han seleccionado para el diseño y construcción del software, puesto que estas decisiones iniciales influyen enormemente en el resto del proceso (especialmente la selección del lenguaje de programación). A continuación, se definen los *stakeholders* (personas con intereses en el sistema) y las cuestiones de diseño, como marca el estándar antes mencionado, lo cual da pie a completar las dos últimas secciones del documento: los puntos de vista, los cuales son 'metamodelos' que responden a unas cuestiones de diseño u otras de las mencionadas anteriormente, y que determinan los requisitos de cómo se debe diseñar para poder mostrar adecuadamente la información de la cuestión a la que pretenden servir; y las vistas, que son las 'implementaciones' de los puntos de vista, haciendo uso de modelos y diagramas gráficos, junto con descripción textual, para cubrir los elementos a los que hacen referencia las cuestiones de diseño de sus respectivos puntos de vista.

Nótese que el diseño aquí descrito es un diseño 'a posteriori'. Es decir, para el desarrollo de la herramienta, tras la extracción de requisitos, se llevó a cabo un proceso de diseño que asegurara una buena construcción posterior del software. Pero dicho diseño se realizó a alto nivel, es decir, sin especificar una gran cantidad de detalles de cómo sería la implementación exacta del código, pero que sirviera de guía a la hora crear la arquitectura del sistema, la organización de sus componentes, el diseño de sus comunicaciones, etc.

En cambio, el diseño que se puede ver aquí incluye un nivel de detalle bastante mayor, lo cual implica que, a partir del diseño inicial, éste se ha completado con el resultado final de la construcción propia del código, ofreciendo un visión completa (mediante vistas de diseño o diagramas) del resultado final del software.

## 8.1 Decisiones de diseño e implementación

Esta sección pretende cubrir, únicamente, aquellas decisiones tomadas con respecto a la selección de herramientas y lenguajes utilizados para modelar e implementar el software

(puesto que las decisiones de diseño propiamente dichas se justifican en aquellos puntos donde aparecen).

Para empezar, en cuanto a las herramientas de diseño, lo más interesante es mencionar que se ha usado el lenguaje de modelado UML 2.5 [17], [18], [19] para las representaciones gráficas de las vistas. Esto es así porque dicho lenguaje cubre, con suficiente detalle, todos los modelados gráficos que describen cada una de las vistas que se van a usar.

Por otro lado, se puede señalar que, para la creación de dichos modelos, se ha hecho uso de la herramienta de modelado 'Modelio v3.2' [20]. Se trata de una herramienta de libre distribución, que permite trabajar con todos los modelos UML que se necesitan, y que resulta de fácil uso y aprendizaje.

En cuanto a la implementación, el lenguaje de programación escogido ha sido Java, en su versión 1.7.0.25. Esto es así para cubrir uno de los requisitos que se le exigían a la herramienta, que es que fuese independiente del sistema operativo en que se creara, pudiéndose ejecutar el código tanto en un sistema Windows como en uno Linux, sin necesidad de recompilarlo. Este requisito lo cumple sobradamente Java.

Otro motivo importante para la elección de este lenguaje es que dispone de una enorme cantidad de bibliotecas diseñadas para ser usadas en él y que facilitan enormemente la tarea de programar, ya sean las bibliotecas propias de la API de Java, como bibliotecas externas creadas por otras personas u organizaciones. De este modo, la posibilidad de reutilización de código ya creado y probado agiliza enormemente la implementación del software y facilita el aseguramiento de su calidad.

Además del lenguaje, se ha escogido un IDE para dar soporte al proceso de implementación del código: NetBeans [21], versión 7.4. Actualmente, utilizar un IDE para el desarrollo del código es una cuestión casi obligatoria para que esta tarea se pueda llevar a cabo en un tiempo razonable y con suficiente calidad. En este ámbito, NetBeans es uno de los IDEs más conocidos y completos para el desarrollo de aplicaciones, especialmente en Java. Ha sido escogido por la madurez de dicha herramienta, que la hace ser realmente completa, con una amplia cantidad y variedad de funciones, y muy intuitiva y fácil de utilizar.

## 8.2 Stakeholders y cuestiones de diseño

### 8.2.1 Stakeholders

Los *stakeholders* son aquellas personas u organizaciones que tienen algún tipo de interés en el diseño del software. Para esta herramienta, los *stakeholders* encontrados son:

- El autor de este PFM. En este caso, es el autor del actual proyecto el que va a realizar todas las fases del desarrollo de la herramienta (extracción y análisis de

requisitos, diseño, implementación, pruebas…) por lo tanto es uno de los principales interesados en todo su desarrollo.

- Los tutores de este PFM. Se puede decir que los tutores de este proyecto hace las veces de cliente, y establecen los requisitos y objetivos que se deben alcanzar en el proyecto, así como validar que el resultado final se adhiere a estos.
- Innotech. Los desarrolladores que crearon el sistema Hydra de gestión de entornos cloud, puesto que esta herramienta mejora y amplia las prestaciones de dicho sistema.
- Posibles desarrolladores futuros. Aquellos desarrolladores que, en un futuro, deseen llevar a cabo tareas de mantenimiento y/o evolución de la herramienta, deben ser tenidos en cuenta y considerar sus posibles necesidades.

### 8.2.2 Cuestiones de diseño

Las cuestiones de diseño (*design concerns*) son áreas de interés con respecto a un diseño software. Es a partir de las cuestiones de diseño que seleccionemos que se usan unos puntos de vista u otros. Para el diseño de esta herramienta, nos vamos a centrar en las siguientes áreas o cuestiones de diseño:

1. Definición de los servicios ofrecidos por el sistema a los usuarios, así como la identificación de dichos usuarios y sus tipos.
2. Composición y despliegue de los elementos físicos del sistema, y relaciones o comunicaciones necesarias entre ellos.
3. Composición modular interna y arquitectura software.
4. Estructura lógica estática (clases, interfaces y sus relaciones) así como la reutilización o implementación de tipos de datos.
5. Comportamiento dinámico del sistema en tiempo de ejecución.

## 8.3 Puntos de vista del sistema

Los puntos de vista que se describen a continuación pretenden marcar las pautas de diseño que se deben seguir (y que se materializan en las vistas) para poder mostrar todos los elementos concernientes a las cuestiones de diseño antes expuestas, para así asegurar una buena implementación de estos durante la fase de codificación.

### 8.3.1 Punto de vista de contexto

"*El punto de vista de contexto representa los servicios prestados por un sujeto de diseño con referencia a un contexto explícito. Ese contexto se define por referencia a los actores, que incluyen usuarios y otros stakeholders, que interactúan con el sujeto de diseño en su entorno. El punto de vista contextual ofrece una perspectiva de "caja negra" del sujeto de diseño.*" (Std. IEEE 1016-2009) [16]

Este punto de vista pretende mostrar las situaciones en las cuales los usuarios del sistema interactúan con éste, de modo que reciben de él algún tipo de valor en respuesta a su petición o interacción, pero sin describir cómo el sistema ha sido capaz de generar la salida.

Así, la finalidad del punto de vista de contexto es el identificar los servicios ofrecidos por el sistema, quiénes son sus actores (los usuarios u otros *stakeholders* capaces de interactuar con cada servicio) y establecer los límites y alcance del funcionamiento y uso del sistema.

Para la descripción gráfica de la vista correspondiente, se van a utilizar los diagramas de casos de uso, que cumplen a la perfección las características necesarias para abordar las cuestiones de diseño de este punto de vista.

### 8.3.2 Punto de vista de composición

"*El punto de vista de composición describe la forma en que el sujeto de diseño es (recursivamente) estructurado en sus partes constituyentes y establece los roles de esas partes.*" (Std. IEEE 1016-2009) [16]

Este punto de vista puede ser refinado en dos vertientes: la lógica, que hace referencia a su descomposición en los módulos lógicos que determinarán la implementación interna del código; y la física, referida a los archivos y elementos físicos que compondrán el sistema, y su despliegue y organización sobre otros dispositivos físicos con capacidad de ejecutarlos y/o almacenarlos.

Las cuestiones de diseño que se quieren abordar en este punto de vista son la estructura de ficheros que van a conformar el sistema completo, su despliegue sobre el equipo y las características de éste, y la estructura necesaria para la comunicación entre la herramienta y el sistema Hydra.

Por tanto, de las dos posibilidades, la única que interesa aquí es la de la composición física. Y para esto, en la vista correspondiente se va a utilizar como representación gráfica los modelos de despliegue, que permiten mostrar los elementos y archivos físicos de que se compone el software y su reparto o despliegue entre nodos.

### 8.3.3 Punto de vista de dependencia

"*El punto de vista de dependencia especifica las relaciones de interconexión y acceso entre entidades. Estas relaciones incluyen información compartida, orden de ejecución o parametrización de interfaces.*" (Std. IEEE 1016-2009) [16]

La vista de dependencia provee de una visión general del sujeto de diseño que permite determinar el impacto de los cambios en los requerimientos o el diseño. Así, los elementos que se muestran aquí son los componentes de diseño de alto nivel que conforman la arquitectura del software, así como sus relaciones y dependencias mutuas.

Existen varios modelos gráficos que permiten representar los conceptos de diseño aquí mencionados. En este caso, se ha decidido apostar por una versión modificada del diagrama de paquetes, en el cual se podrá mostrar la estructura de capas del diseño software. De este modo, se va a representar en él la arquitectura por capas creada para la herramienta, así como el conjunto de paquetes que forma cada capa y las relaciones y dependencias entre ellas.

### 8.3.4 Punto de vista lógico

"*El propósito del punto de vista lógico es el elaborar los tipos existentes y diseñados, y sus implementaciones como clases e interfaces con su estructura de relaciones estáticas.*" (Std. IEEE 1016-2009) [16]

Este punto de vista, en cierto modo, nos aporta la visión que nos podría aportar el punto de vista de composición lógico (y por eso, sólo se tenía en cuenta el tipo físico de los dos posibles). Aunque en este caso, el nivel de abstracción en la vista de composición es algo mayor, puesto que no se abordan las mismas cuestiones de diseño.

Así, el punto de vista lógico se centra en el desarrollo y reutilización de las abstracciones adecuadas y sus implementaciones. Para cualquier plataforma de implementación, tenemos disponibles una serie de abstracciones en relación al sujeto de diseño, las cuales habrá que considerar si usar, redefinir o definir otras nuevas en términos de los tipos existentes. La cuestión de diseño principal, por tanto, es la elección de las abstracciones que se van a utilizar, el diseño de las nuevas que se necesiten, y las relaciones estáticas entre todas ellas.

De este modo, la representación gráfica más comúnmente usada en la vista correspondiente a este punto de vista (ya que es la que mejor encaja para responder a las cuestiones de diseño citadas), y que también va a ser utilizada aquí, es el diagrama de clases.

### 8.3.5 Punto de vista de dinámica de estados

"*Este punto de vista es de interés para aquellos sistemas que son reactivos ante diversos eventos, o cuyo comportamiento interno es de interés en el diseño del mismo.*" (Std. IEEE 1016-2009) [16]

El resto de puntos de vista mencionados (excepto el de contexto) nos aportan una visión de la estructura estática de los elementos que componen el sistema y sus relaciones, a distintos niveles de abstracción. Este último punto de vista pretende introducir una visión 'dinámica' del sistema, es decir, mostrar el comportamiento que éste va a tener durante su ejecución.

Así, lo que en este punto de vista se quiere mostrar es la lógica de ejecución del sistema, el flujo de control que determina cómo realiza sus funciones. E igualmente, se abordan cuestiones como la eficiencia del sistema a la hora de trabajar.

Existen diversos modelos y diagramas que se pueden utilizar para esquematizar el contenido de la vista de este punto de vista, como los diagramas de comunicación o los de diagramas de secuencia. En estos dos casos, el nivel de detalle alcanzable es muy alto, pudiendo mostrar explícitamente cómo las clases construidas interaccionan entre ellas intercambiando mensajes, o las distintas estructuras de control utilizadas (alternativas, iterativas, etc.). Esto sería prácticamente como escribir el pseudocódigo de la herramienta, y ésta no es la finalidad de este documento.

Así pues, se pretende mostrar simplemente una visión general de cómo se ha decidido que el programa trabaje en su totalidad, sin bajar demasiado el nivel de abstracción. Por lo tanto, se ha elegido como diagrama para exponer esta información el diagrama de actividades, el cual nos permite mostrar las 'actividades que se llevan a cabo' de forma abstracta, sin entrar en detalle de si internamente dicha actividad o acción se corresponde con un método de una clase, o con un trozo de código más amplio que implica la llamada de más de un método.

## 8.4 Vistas del sistema

### 8.4.1 Vista de contexto

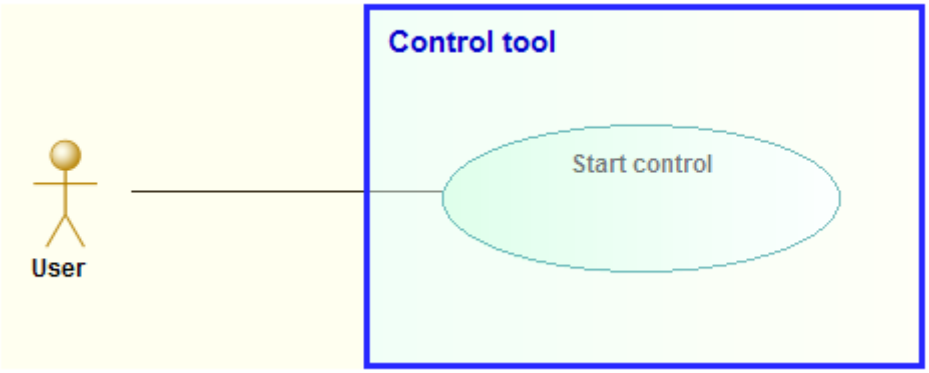


Diagrama 1, Diagrama de casos de uso

Para esta herramienta no existe diferenciación entre tipos de usuario. Por lo tanto, sólo hay un actor necesario, que hace las veces de usuario, administrador o cualquier otro rol que imaginemos.

Además, este usuario únicamente puede llevar a cabo una acción directa sobre la herramienta, que es iniciarla. A partir de ahí, el software iniciará automáticamente su labor, sin que el usuario tenga que volver a interactuar directamente con él.

Los resultados del desempeño de la herramienta son devueltos al usuario no de manera directa, sino a través de los archivos de registro que están a disposición de éste para su consulta cuando estime oportuno.

Y por otro lado, la interacción que sí puede llevar a cabo el usuario para modificar ciertos comportamientos de la herramienta (aquellos que se refieren a las opciones configurables), de nuevo tampoco se realiza de forma directa, sino a través del archivo de configuración, el cual, el usuario puede modificar cuando quiera, y el software leerá a continuación.

Es por todo esto que, siendo un software tan autónomo y que requiere de tan poca atención por parte del usuario, el diagrama de casos de uso de esta vista de contexto queda tan sencillo.

### 8.4.2 Vista de composición

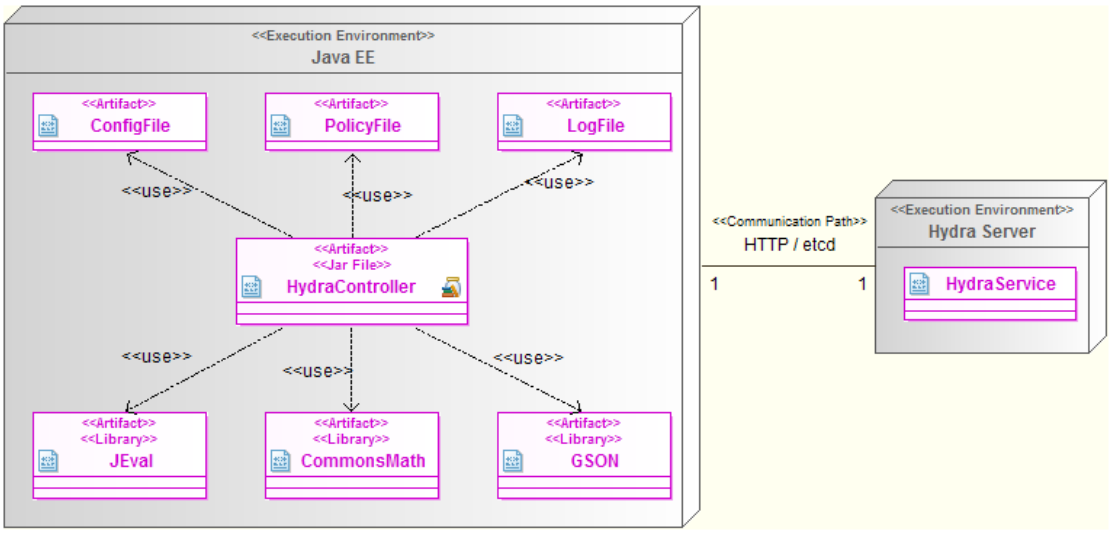


**Diagrama 2, Diagrama de despliegue**

La característica principal de la herramienta en esta vista, es el hecho de que existan dos nodos diferentes los cuales deben poder comunicarse entre sí.

En el primero de ellos (*Java EE*), se ha desplegado el sistema de control de Hydra, que está compuesto por un archivo principal ejecutable .jar (*HydraController*), el cual contiene el código compilado para ejecutar la aplicación. Es necesario, por tanto, que en este mismo nodo se ubiquen las bibliotecas de que hará uso (*JEval*, *CommonsMath* y *GSON*), así como los dos archivos de texto a los que tiene que acceder para obtener la información de configuración y las políticas (*ConfigFile* y *PolicyFile*, respectivamente). Por último, también se ubicará en este mismo nodo el archivo de registro (*LogFile*) donde la herramienta vuelca la información sobre su ejecución, ya que es la propia herramienta la que decide donde se ubica, y siempre lo hará en su mismo nodo.

En el segundo nodo (*Hydra Server*), el único elemento que nos interesa resaltar es el servicio de Hydra (*HydraService*). Éste es un servicio que debe estar ejecutándose en un servidor, preparado para escuchar peticiones a través de los puertos que tenga definidos.

Los nombres de estos elementos aquí indicados no se corresponden con los nombres reales finales que tendrán los archivos físicos una vez desplegados.

Para que la herramienta pueda llevar a cabo su labor, el artefacto HydraController debe ser capaz de comunicarse con HydraService. Para ello, ambos nodos deben disponer de una ruta de comunicación que les permita intercambiar mensajes entre sí utilizando el protocolo HTTP, puesto que es este protocolo, junto con el uso de etcd, la forma en que se ha prescrito que se van a comunicar los dos artefactos antes mencionados.

En cuanto a los nodos, hay que notar que no se están especificando dos dispositivos físicos como tal, como un equipo o un servidor, sino que ambos señalan que son entornos de ejecución.

El primero, es la plataforma de ejecución Java, que permite tanto el desarrollo como la ejecución de aplicaciones desarrolladas en lenguaje de programación Java. Esto asegura que dicho entorno disponga de dos de los elementos fundamentales para la ejecución del archivo .jar HydraController: la máquina virtual de Java (del tipo que sea, según el sistema operativo en que nos encontremos) y la API de Java (con todas sus bibliotecas de clases y funciones, a las algunas de las cuales la aplicación necesita acceder, lo cual se ve en más detalle en el siguiente apartado).

El segundo, se trata de un servidor capaz de ejecutar Hydra. En este caso, la descripción de las características de este nodo no es tan relevante en el diseño de esta herramienta, puesto que no le afectan, sino que simplemente deben conocerse por aquellos usuarios o administradores que deben desplegar y ejecutar el servicio.

Y llegados a este punto, quizás lo más interesante a señalar, y que se deriva de lo expuesto arriba (que los nodos son entornos de ejecución y no dispositivos físicos) es que dichos nodos pueden encontrarse desplegados en nodos físicos diferentes, es decir, que estén ubicados en servidores distintos y tengan que comunicarse a través de una infraestructura de red de comunicaciones; o que estén ubicados en el mismo servidor, por lo que realmente los mensajes que se mandarían los artefactos de un nodo a otro no tendrían que salir del propio equipo servidor.

Pero, en cualquier caso, no cambia el diseño del despliegue, puesto que cada artefacto necesita que el nodo físico en que se despliegue cumpla con los requisitos de los entornos de ejecución especificados para cada uno, ya sea un mismo equipo, o en equipos diferentes. Y esto tampoco afecta al método de comunicación, el cual, en cualquiera de los dos casos, seguirá realizándose a través del protocolo HTTP con el formato etcd.

#### *8.4.2.1 Artefactos*

Tenemos los siguientes artefactos que componen el diagrama de despliegue:

- **HydraController**. Archivo .jar. Código fuente compilado con la implementación de la herramienta, listo para ser ejecutado.
- **JEval**. Archivo .jar. Biblioteca externa JEval.
- **CommonsMath**. Archivo .jar. Biblioteca externa Commons Math.
- **GSON**. Archivo .jar. Biblioteca externa GSON.
- **ConfigFile**. Fichero de texto. En formato de texto plano, con las opciones de configuración de la herramienta y sus valores creados por el usuario.
- **PolicyFile**. Fichero de texto. En formato JSON, con la descripción de las políticas de balanceo.
- **LogFile**. Fichero de texto. En el cual, la herramienta registra los eventos que se produzcan.
- **HydraService**. Servicio o proceso en ejecución. Debe estar ejecutándose y escuchando en un puerto de su servidor para poder recibir los mensajes de HydraController.

### 8.4.2.2 Nodos

Tenemos los siguientes nodos que componen el diagrama de despliegue:

- **Java EE**. El entorno o plataforma de ejecución característico de Java, que incluye, entre otras cosas, la máquina virtual Java y el conjunto de bibliotecas que conforman su API, configurado para poder ejecutar archivos .jar.
- **Hydra Server**. El entorno de ejecución que permite desplegar y ejecutar el servicio de gestión de entornos cloud Hydra.

## 8.4.3 Vista de dependencia

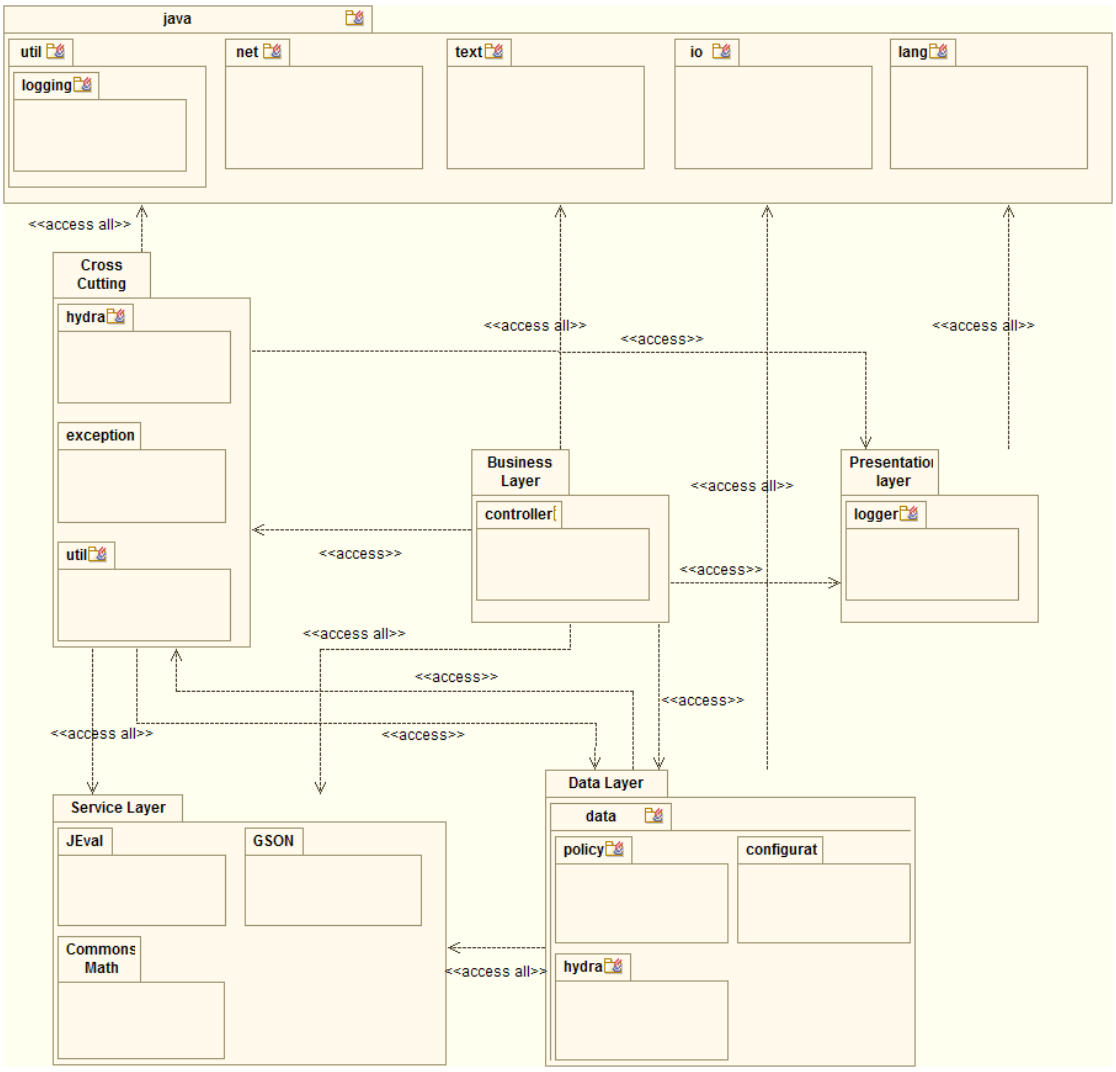


Diagrama 3, Diagrama de capas/paquetes

### 8.4.3.1 Capa de negocio

La capa de negocio se corresponde con la capa del mismo nombre, o capa controlador, dentro de la arquitectura MVC. Contiene el código que conforma la estructura y comportamiento lógico de la herramienta (es decir, determina el flujo de control principal). Así, responde a las acciones y eventos que se solicitan o tienen lugar (por lo general, a través de la capa de presentación), invoca peticiones sobre la capa de datos para que se lleven a cabo sobre ellos las operaciones necesarias que se han solicitado y,

finalmente, actualiza la presentación de esos datos y sus cambios, de nuevo en la capa de presentación. De este modo, se puede entender la capa de negocio o controlador como un intermediario entre los datos y la forma de su visualización (es decir, entre las otras dos capas que definen el modelo MVC, el 'modelo' y la 'vista').

En este caso, lo mayoría de los eventos a los que va a responder el controlador vienen determinados internamente, y no por las acciones de un usuario (como por ejemplo, la finalización del tiempo de espera entre ciclos de ejecución, que consecuentemente, marca el inicio de uno nuevo), de modo que sólo tendrá la necesidad de comunicarse con la capa de presentación para mostrar resultados de la ejecución.

Esta capa, por sus características, tiene la necesidad de comunicarse, usar y/o importar componentes y clases de todas las demás capas de que se compone la herramienta, para así controlar toda la ejecución.

Está formada por un único paquete:

- **controller**. Contiene las clases referidas al control principal de la lógica de ejecución del programa.

#### *8.4.3.2 Capa de presentación*

La capa de presentación se corresponde con la capa del mismo nombre, o capa vista, en el ya mencionado modelo arquitectónico MVC. Ésta se encarga de la presentación de la información de la capa de datos mediante un formato adecuado para su visualización por parte del usuario (es decir, haciendo uso de la interfaz de usuario creada para la ocasión), permitiendo a su vez a éste, la interacción y modificación de dicha información.

El acceso a los datos, para poder mostrarlos, por parte de esta capa no se hace de manera directa, sino que se efectúa a través de la capa de datos, pidiéndole a ésta los valores actualizados de los datos.

En el caso de esta herramienta, no hay una capa de presentación en el sentido más clásico en que ésta se entiende en el modelo MVC, puesto que no hay un usuario que esté constantemente observando y/o interactuando con ella, y por tanto, la interfaz de usuario es muy limitada.

A pesar de ello, sí que se da una presentación de información al usuario, que se produce a través de una salida de información tanto a un archivo de registro, como a la consola del sistema. Esta función será la que conforme la capa de presentación.

Está formada por un único paquete:

- **logger**. Contiene las clases que controlan la salida de información a través de la consola del sistema y del archivo de registro, así como el formato concreto de dicha información al ser mostrada.

#### 8.4.3.3 Capa de datos

La capa de datos se corresponde con la capa del mismo nombre, o capa modelo, en el renombrado modelo de arquitectura software MVC. Se encarga de establecer los permisos y la forma concreta de los accesos a los datos, y está constituida por las estructuras de datos que dan el soporte a toda la información que maneja el sistema, así como las operaciones que permiten actualizar dicha información y ser visualizada (internamente en el sistema, no por parte del usuario, que eso ya se ha mencionado que lo realiza la capa de presentación).

Está formada por los siguientes paquetes:

- **data**. Contiene las clases que soportan datos que son comunes a toda la herramienta (valores constantes y/o estáticos de acceso y ámbito general en la herramienta).
- **data.configuration**. Contiene las clases que conforman las estructuras de datos y operaciones que soportan la información configurable de la herramienta, y que es leída del archivo de configuración.
- **data.policy**. Contiene las clases que conforman las estructuras de datos y operaciones que soportan la información de las políticas de balanceo, y que es leída del archivo de políticas.
- **data.hydra**. Contiene las clases que conforman las estructuras de datos y operaciones que soportan la información de monitorización del entorno cloud gestionado por el sistema Hydra, y que es enviada desde éste.

#### 8.4.3.4 Capa cruzada

La capa cruzada es una capa creada para esta ocasión, la cual contiene componentes, clases u operaciones que son de uso común en toda la herramienta, y que resultan conceptualmente más difíciles de introducir en alguna de las otras capas del modelo MVC. Es por lo tanto, una capa auxiliar, la cual puede ser accedida o usada por los componentes de cualquiera de las demás capas.

Está compuesta por los siguientes paquetes:

- **exception**. Contiene las clases que determinan las excepciones únicas y personalizadas creadas para para ser reconocidas y manejadas en el ámbito de esta herramienta.
- **hydra**. Contiene las clases que sirven, a modo de biblioteca de funciones, para llevar a cabo todas las comunicaciones e intercambios de información con el sistema Hydra.
- **util**. Contiene una serie de clases auxiliares, que tienen en común que son usadas o referenciadas desde distintos puntos del código para utilizar sus funciones o atributos cuando son necesarios.

#### 8.4.3.5 Capa de servicio

Esta capa no forma parte realmente de la implementación propia de la herramienta, pero es mostrada debido al interés de mostrar las dependencias que ocurren con el resto del

sistema. Está compuesta por bibliotecas externas que dan soporte a algunas de las funciones que son necesarias efectuar durante la ejecución, lo cual hace que tengan una función de servicios externos (de ahí el nombre de la capa).

Está formada por las siguientes bibliotecas externas:

- **JEval**. Se trata de una biblioteca que permite la evaluación de expresiones matemáticas y lógicas, la cual es utilizada en la herramienta durante el análisis de las condiciones de los estados de las aplicaciones incluidas en las políticas de balanceo.
- **GSON**. Es una biblioteca de código abierto, creada por Google, que permite la serialización y deserialización entre objetos Java y su representación en notación JSON. De este modo, permite leer y acceder de manera sencilla a datos en formato JSON, transformándolos automáticamente en objetos Java, así como permite una forma sencilla de crear nuevos objetos de este tipo y convertirlos automáticamente en documentos JSON. Esta biblioteca es utilizada para poder interpretar los datos enviados por Hydra, así como el archivo que contiene las políticas de balanceo.
- **Commons Math**. Se trata de una biblioteca, creada por Apache Software Foundation, que contiene gran cantidad de clases que permiten realizar todo tipo de operaciones matemáticas y estadísticas, las cuales no son posibles de efectuar haciendo uso exclusivamente de la API de Java. Es utilizada para llevar a cabo las operaciones estadísticas definidas en la herramienta sobre los valores en tiempo real de las variables de monitorización del sistema cloud de Hydra.

#### *8.4.3.6 Capa API de Java*

Al igual que la anterior, los elementos de esta capa realmente no forman parte del código de la herramienta, pero tiene su interés mostrarla en esta vista para visualizar sus dependencias con el resto de la aplicación. Como su nombre indica, esta capa se corresponde con la API de Java, y está formada por el conjunto de clases y/o paquetes que son accedidos, usados e importados a lo largo de la herramienta.

En este caso, no se va a entrar a enumerar y describir los paquetes y clases del API de Java que son utilizados, puesto que no se considera de verdadero interés para la compresión del diseño de la aplicación. Se considera suficiente con mostrarlos en el diagrama de la vista actual, y en los diagramas de la vista lógica.

## 8.4.4 Vista lógica

### *8.4.4.1 Capa de negocio*

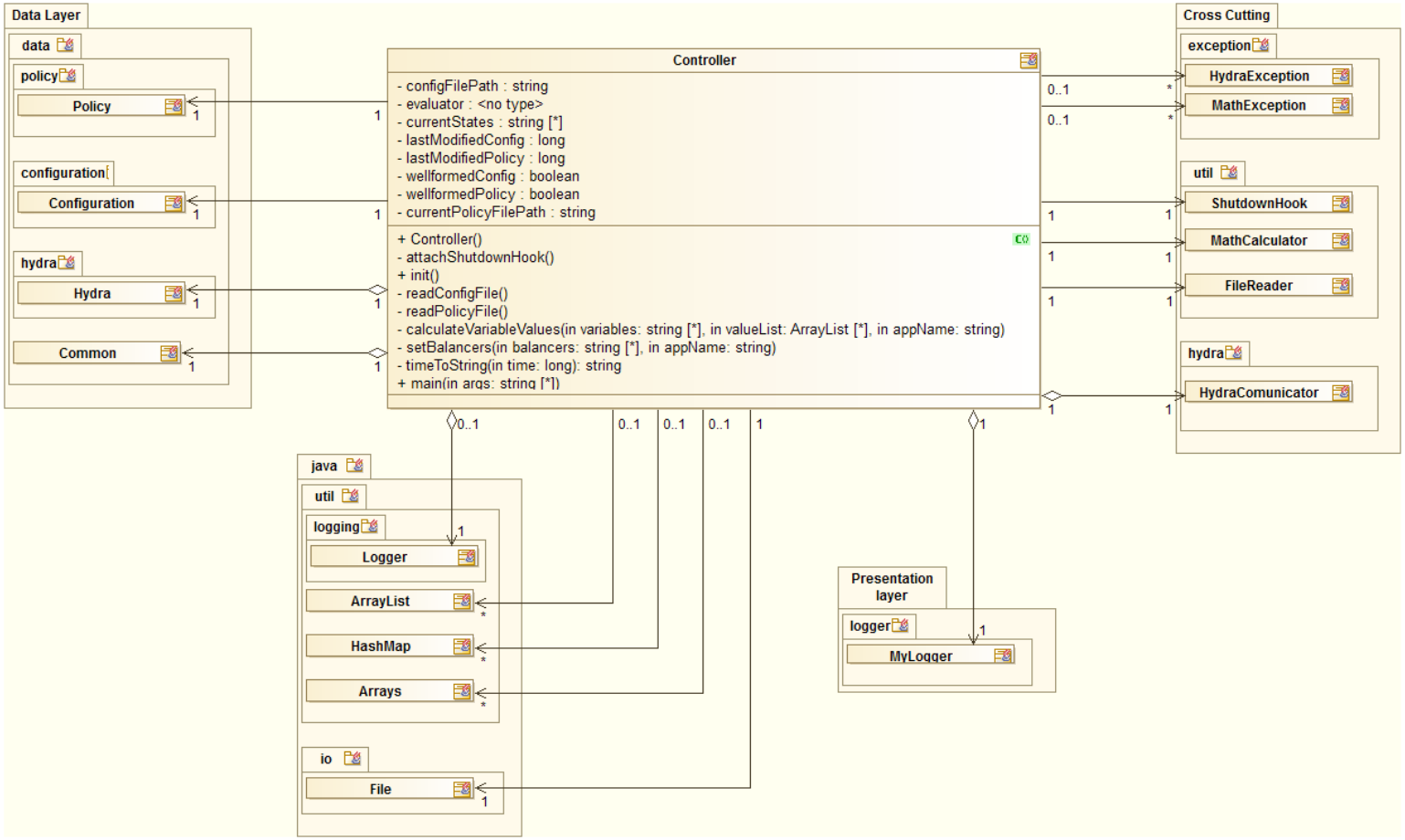


Diagrama 4, Diagrama de clases de la capa de negocio

### Controller

Clase principal que inicializa y controla la ejecución de los ciclos que se deben realizar para la monitorización, reconfiguración y control de Hydra.

En ella, la lógica de la ejecución se realiza por ciclos, y tras cada uno de estos, la ejecución se bloquea durante un tiempo determinado antes de pasar al siguiente. Básicamente, los pasos que se ejecutan en cada ciclo son:

- Se lee el fichero con las opciones de configuración.
- Se lee el fichero JSON con la definición de las políticas y se analiza.
- Se solicita al sistema Hydra la información de monitorización actual de las aplicaciones y se analiza.
- Se determina, aplicación por aplicación, el estado actual de cada una de ellas en base a sus datos de monitorización y las condiciones de las políticas.
- Finalmente, se modifican los balanceadores en Hydra según el estado de cada aplicación.

A modo de optimización, los ficheros de configuración y de políticas sólo se leen en el caso de que hayan sido modificados. Además, se almacena el estado en que ha quedado cada aplicación, y sólo se modifican los balanceadores en Hydra cuando cambia el estado de las aplicaciones.

### 8.4.4.2 Capa de presentación

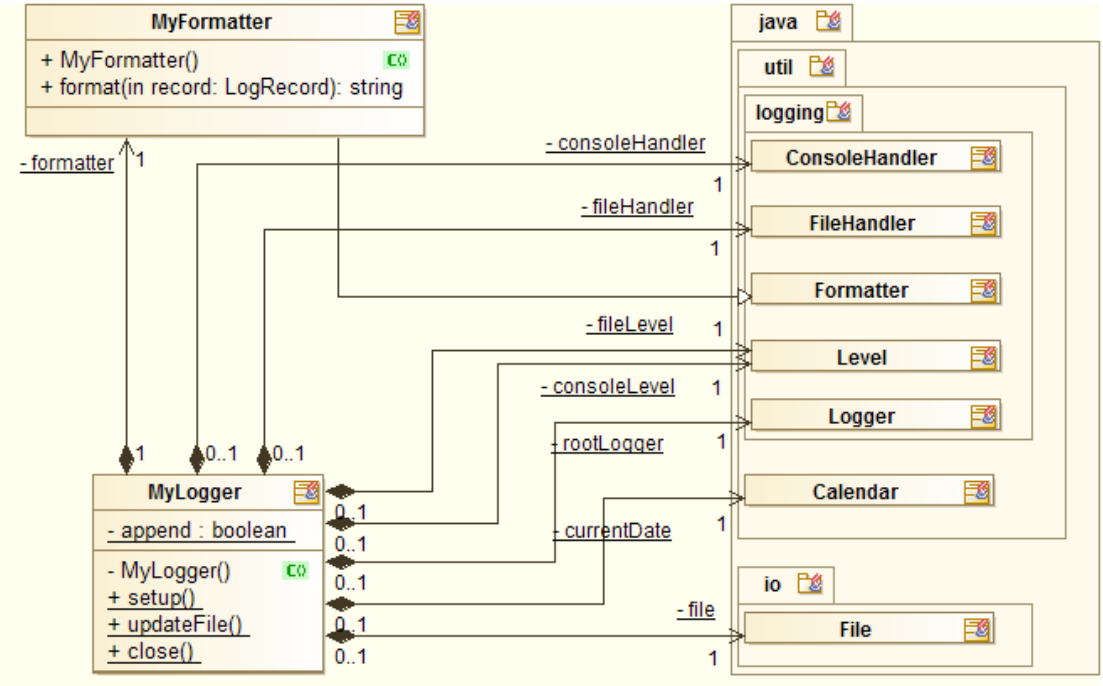


Diagrama 5, Diagrama de clases de la capa de presentación

#### MyFormatter

Establece un formato para la salida de datos que se produce tanto a través de la consola, como su escritura en el archivo de registro.

#### MyLogger

Definición de una clase registro específica para la aplicación. Este logger hereda de la clase Logger propia de Java, y tiene dos salidas, una por pantalla y otra a un fichero de registro. Define, para cada una, un nivel de importancia o valor que determina el tipo de mensajes que van a mostrar ambas.

Además de mostrar los datos, también se encarga de gestionar la creación y mantenimiento del fichero del registro. Para ello, usa siempre un fichero que incluye en el nombre del mismo la fecha actual de los eventos registrados, ya sea creándolo o añadiendo nueva información al final de éste. Cuenta, así mismo, con una función que permite revisar la fecha actual, y, si es necesario, cerrar el fichero actual y abrir uno nuevo con la fecha nueva.

### 8.4.4.3 Capa de datos

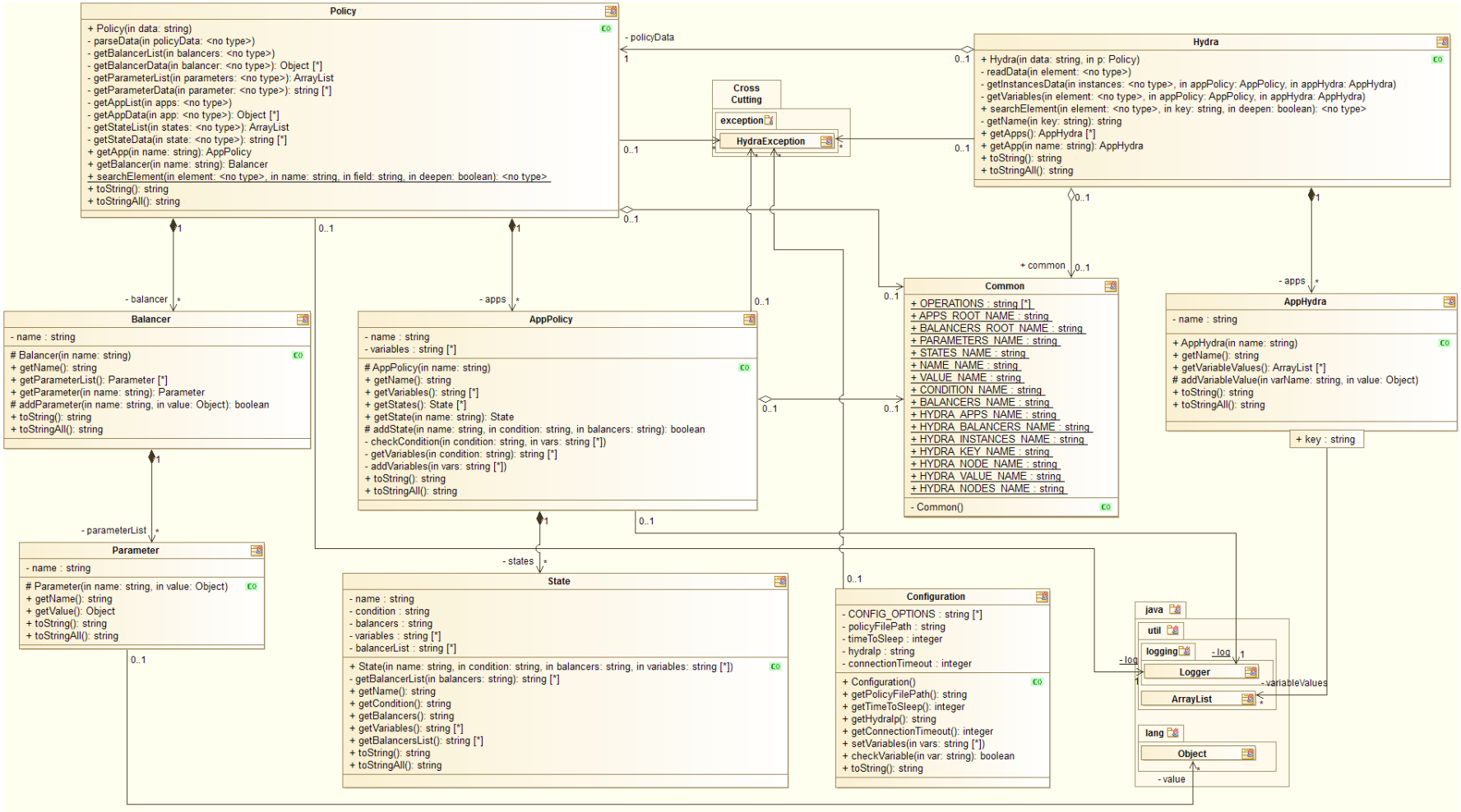


Diagrama 6, Diagrama de clases de la capa de datos

### Common
Contiene un conjunto de variables constantes estáticas, cuyos valores son consultados en diversas partes del código. Es una clase no instanciable.

### Configuration
Esta clase soporta las estructuras de datos y las funciones que permiten almacenar y consultar los valores de las opciones de configuración de la herramienta, así como actualizar dichos valores cuando otros nuevos son leídos del archivo de configuración.

### Policy
Esta es la clase principal (o raíz) que soporta las estructuras de datos y las funciones que permiten almacenar, tratar y consultar los datos incluidos en el fichero de políticas.

### AppPolicy
Esta clase contiene las estructuras de datos y las funciones que permiten almacenar, tratar y consultar los datos del fichero de políticas referentes a cada aplicación, junto con sus estados.

### State
Esta clase contiene las estructuras de datos y las funciones que permiten almacenar y consultar los datos del fichero de políticas referentes a cada estado de las aplicaciones.

### Balancer
Esta clase contiene las estructuras de datos y las funciones que permiten almacenar, tratar y consultar los datos del fichero de políticas referentes a cada balanceador, junto con sus parámetros.

### Parameter
Esta clase contiene las estructuras de datos y las funciones que permiten almacenar y consultar los datos del fichero de políticas referentes a cada parámetro de los balanceadores.

### Hydra
Esta es la clase principal (o raíz) que soporta las estructuras de datos y las funciones que permiten almacenar, tratar y consultar los datos enviados por el sistema Hydra referentes al conjunto de todas sus aplicaciones.

### AppHydra
Esta clase contiene las estructuras de datos y las funciones que permiten almacenar, tratar y consultar los datos enviados por el sistema Hydra referentes a cada aplicación (es decir, sus instancias y balanceadores).

### 8.4.4.4 Capa cruzada

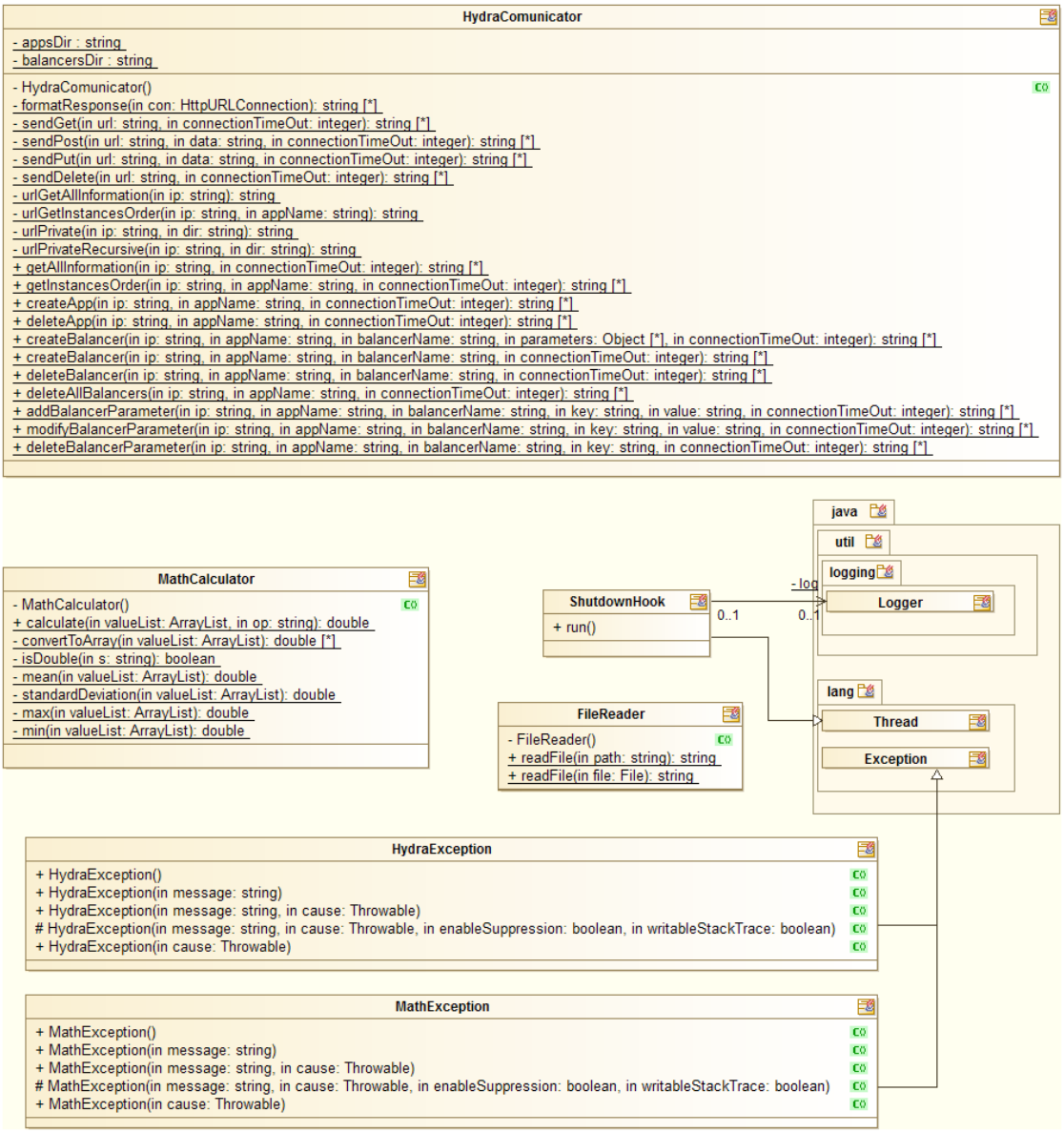


**Diagrama 7, Diagrama de clases de la capa cruzada**

#### HydraComunicator

La clase HydraComunicator contiene un conjunto de funciones que permiten la comunicación con el servicio Hydra de gestión de entornos cloud, a través de una amplia variedad de solicitudes y órdenes. Permite tanto obtener información por parte de Hydra, como modificar su configuración.

Para todas las funciones, se especifica siempre un límite máximo de espera para acotar el tiempo para establecer la conexión, y la dirección IP a donde se deben enviar las solicitudes (es decir, la dirección del servidor en que se está ejecutando Hydra).

Se trata enteramente de una clase auxiliar estática, es decir, es una clase no instanciable donde todos sus miembros (atributos y métodos) son estáticos.

### HydraException

Indica que se ha producido algún tipo de error durante la ejecución del programa, debido a errores o situaciones propias de éste, como que no se haya podido conectar con Hydra, que no exista un fichero de políticas o esté mal construido, etc.

La clase hereda de la clase Exception de Java.

### MathException

Indica que se ha producido algún tipo de error durante la ejecución de alguna de las operaciones estadísticas que se realizan sobre los valores de las variables de monitorización de Hydra. Se requiere que este tipo de errores se distinga de los errores generales que muestra la clase 'HydraException'.

La clase hereda igualmente de la clase Exception de Java.

### MathCalculator

Clase que implementa las operaciones y cálculos matemáticos que se deben realizar sobre los valores de las variables de monitorización del entorno cloud gestionado por Hydra.

Al igual que la clase 'HydraComunicator', es una clase estática no instanciable.

### FileReader

Se encarga de abrir, leer y cerrar ficheros de texto, y de devolver su contenido como una única cadena de caracteres. Concretamente, es utilizada para acceder a la información de los dos ficheros que se manejan en esta herramienta (sin contar el fichero de registros): el fichero de configuración y el de políticas.

Nuevamente, se trata de una clase estática no instanciable.

### ShutdownHook

El contenido de esta clase se ejecuta durante la finalización de la ejecución del programa, y sirve para realizar tareas que deben ser hechas justo en ese momento.

Es un requisito de la forma de trabajar de la API de Java con los ficheros de registro, y se encarga, concretamente, de asegurarse de cerrar correctamente el archivo de registro que se esté utilizando en el momento de terminar la ejecución.

### 8.4.5 Vista de dinámica de estados

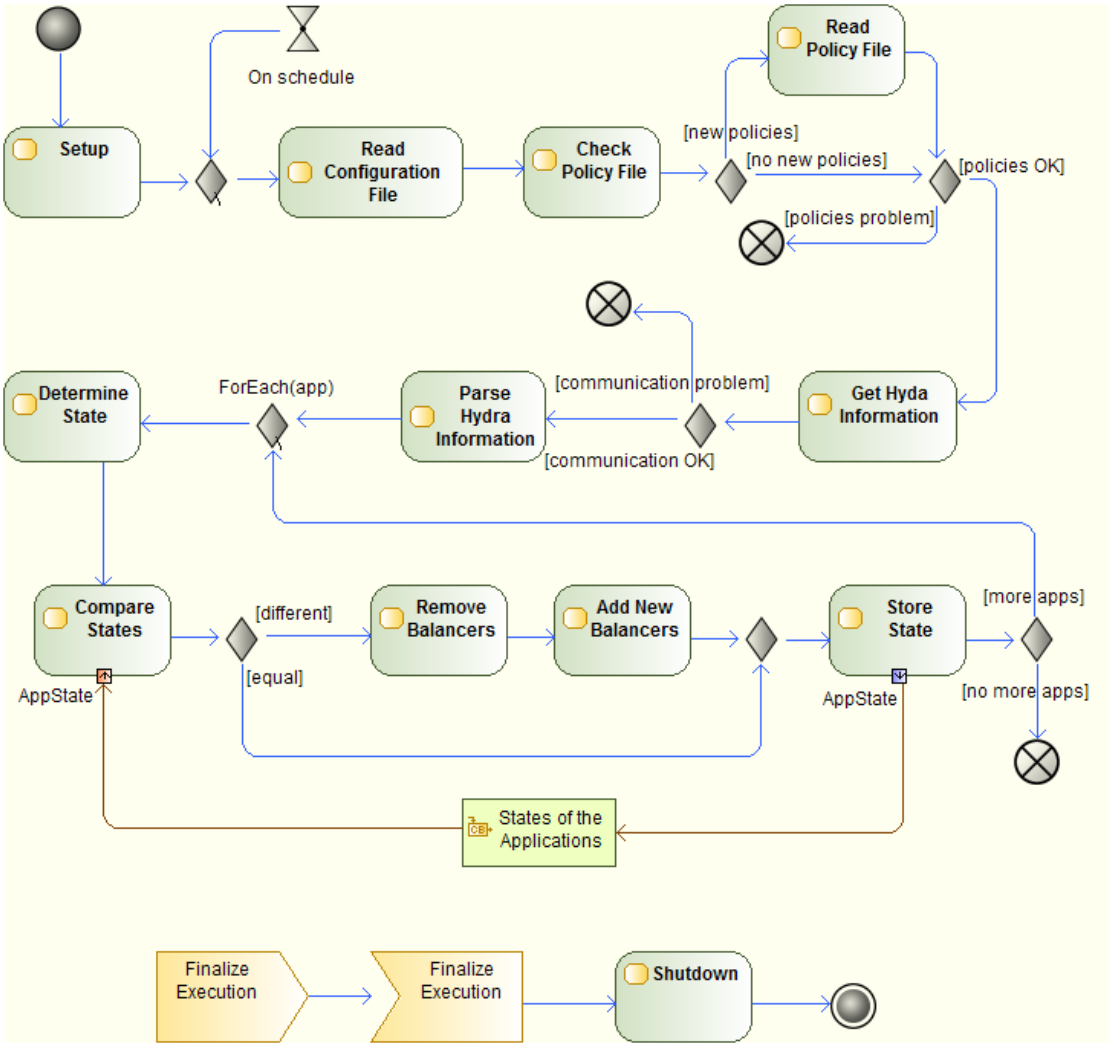


Diagrama 8, Diagrama de actividades

A continuación, la descripción de los pasos del flujo de control que componen la lógica de ejecución:

1. Inicialmente, el servicio es puesto en funcionamiento mediante la acción directa del usuario.
2. El primer paso es inicializar los atributos y las variables que se utilizarán durante la ejecución, entre ellos, el logger (el archivo donde se va a escribir, el formato de escritura, etc.) y una configuración inicial por defecto.
3. A partir de aquí, el programa entra en un ciclo infinito donde realizará periódicamente las operaciones que vienen a continuación. A este punto se llega por primera vez tras la inicialización de los valores anteriores. El resto de las veces, será a través de un evento cuya ejecución está determinada por el tiempo de espera entre ciclos de ejecución.
4. Al iniciar cada ciclo de ejecución, se comprueban, consecutivamente, tres grandes bloques de datos necesarios:

a. El archivo de configuración, el cual, se lee si no se había leído ya (o lo que es lo mismo, si el que había ha sido modificado). Se recogen y se incorporan sus datos al programa si así corresponde. En el caso de que haya algún tipo de problema con el archivo, esto es registrado a través del logger, pero la ejecución puede continuar sin problemas puesto que siempre se dispondrá de un conjunto de datos de configuración correctos (al menos, los de por defecto).
b. El archivo de políticas, el cual se lee si no se había leído ya (o lo que es lo mismo, si el que había leído ha sido modificado). Se recogen y se incorporan sus datos al programa si así corresponde. En el caso de que haya algún tipo de problema con el archivo, esto es registrado a través del logger. A continuación, se puede dar la situación de que ya se disponga de unos datos de políticas leídos con anterioridad. Si es así, el ciclo de ejecución actual puede continuar, pero si no, entonces el ciclo termina.
c. La información de monitorización de Hydra, la cual, se solicita nuevamente al servidor Hydra mediante HTTP en cada ciclo. En el caso de que se produzca algún tipo de fallo durante la comunicación con Hydra, esto se registra a través del logger, y la ejecución del ciclo actual debe terminar. En caso contrario, se puede continuar.

5. Llegados a este punto del ciclo, se dispone de toda la información necesaria para que la aplicación cumpla su cometido. Antes de pasar al siguiente paso, analiza y procesa la información de Hydra para tener un acceso más rápido y fácil a ella posteriormente.
6. Finalmente, nos encontramos con un bucle iterativo, que realiza la siguiente secuencia de pasos para cada una de las aplicaciones sobre las que Hydra haya devuelto datos:
    a. Lo primero es determinar el estado en que se encuentra el cloud para la aplicación que nos concierne. Para ello, es necesario realizar las siguientes tareas:
        i. Agrupar todos los valores de las variables de monitorización de los distintos nodos que ejecutan la aplicación en cuestión.
        ii. Aplicar sobre dichos valores las operaciones estadísticas necesarias, es decir, aquellas que se usan en las condiciones de los estados del archivo de políticas.
        iii. Por último, evaluar una por una las condiciones de los estados mencionados, hasta determinar qué condición se hace cierta y, por tanto, en qué estado se encuentra el cloud para esa aplicación.
    b. Con el estado actual del cloud ya fijado, se compara con el estado en que había quedado en el anterior ciclo de ejecución. Este es un dato que siempre se mantiene de un ciclo al siguiente. Si resulta que es el mismo estado, entonces no es necesario hacer cambios en la configuración de Hydra, por lo que saltamos los siguientes dos puntos. Si el estado ha

cambiado, o éste es el primer ciclo de ejecución y por tanto no tenemos información de estados anteriores, entonces:

i. Se eliminan todos los balanceadores configurados actualmente en Hydra para la aplicación actual.
ii. A continuación, se declaran los nuevos balanceadores que haya descritos en el archivo de políticas para la aplicación y el estado que se ha hecho cierto en este caso.

c. En último lugar, antes de iterar de nuevo con otra aplicación, se guarda el estado del cloud para la aplicación actual, para que en posteriores ejecuciones sirva para no reconfigurar Hydra si el estado del cloud no ha cambiado.

7. Cuando ya no quedan más aplicaciones, el bucle termina, al igual que el flujo de control y, por tanto, el ciclo de ejecución actual.
8. Por otro lado, puesto que el programa se mantiene haciendo ciclo infinitos de ejecución, sólo un evento de finalización, efectuado externamente (por el usuario, el sistema operativo…) puede poner fin a la ejecución del programa. Cuando esto pasa, la aplicación se asegura de cerrar correctamente ciertos elementos (como el archivo de registro) y el proceso termina totalmente.

# 9 Plan de pruebas

Esta sección cubre la fase de diseño y creación de un plan de pruebas para la verificación del software construido.

Para esta sección, se pretendía seguir, como con las anteriores secciones, un modelo estandarizado que sirviera de guía para la estructura de la misma. El modelo aplicable aquí es el '*IEEE Std. 829-2008, IEEE Standard for Software and System Test Documentation*' (Estándar para la Documentación de Pruebas de Software y de Sistemas). Pero en este caso, el modelo describe una metodología que tiene como resultados una muy amplia serie de documentos excesivamente técnicos y extensos, que se salen por completo del objetivo de la memoria del presente proyecto. Por lo tanto, se ha decidido seguir una estructura más 'libre', que se ajusta a las necesidades de este documento, y que pone de manifiesto las pruebas realizadas y el plan seguido, con el nivel de detalle que se precisa.

Concretamente, esta sección se ha estructurado en torno a las tres grandes fases en que se han organizado las pruebas:

- La primera fase comprende las pruebas unitarias realizadas a cada una de las clases por separado, así como una serie de pruebas de integración realizadas sobre todo el software en su conjunto, pero en un entorno 'controlado', es decir, sin usar el sistema Hydra, tan sólo con datos sintéticos de prueba.
- La segunda fase contiene otro grupo de pruebas de integración, donde sí se utiliza un sistema Hydra, aunque 'minimalista', es decir, un sistema pequeño que contiene pocos datos y se compone de pocos nodos.
- Finalmente, la tercera fase de pruebas son otro grupo de pruebas de integración, pero realizadas sobre un sistema cloud real con Hydra ejecutándose en él.

## 9.1 Fase 1

La primera fase de pruebas está compuesta tanto de pruebas unitarias como de integración. La característica en común que tienen todas estas pruebas es que se han realizado en un entorno controlado sin el uso real del sistema Hydra.

Por lo tanto, tanto para las pruebas unitarias como para las de integración, ha sido necesario crear software controlador (*driver*) y de esqueleto (*stub*). Un 'controlador' es un programa principal que recibe los datos de entrada del caso de prueba, los pasa al componente que hay que probar e imprime los resultados. Los 'esqueletos' reemplazan al componente que estamos probando (o a los invocados por éste). Un esqueleto o 'subprograma simulado' usa la interfaz del módulo subordinado, verifica la entrada de datos, realiza una mínima manipulación sobre ellos y devuelve el control al módulo de prueba.

Por otro lado, la otra característica común de esta fase de pruebas es que se ha ido realizando poco a poco en conjunto con la construcción del código de la aplicación. Es decir, a diferencia de las otras fases de pruebas, las cuales se realizaron una vez

finalizada la construcción de todo el código, esta fase se ha llevado a cabo en paralelo al desarrollo del código.

El objetivo principal de esta primera fase es depurar el código de todos los errores y fallos de codificación (*bugs*) posibles. Para ello, se han llevado a cabo una serie de pruebas de caja blanca, las cuales tienen su fundamento en el conocimiento del funcionamiento interno de la aplicación. En ellas, se han tenido en cuenta principalmente dos variables: los datos de entrada, y la metodología para la creación de casos de prueba.

En cuanto a los datos de entrada, puesto que se ha hecho un uso intensivo de los esqueletos a la hora de probar las diferentes clases por separado, se ha intentado que estos fueran lo más variados posibles. Por ejemplo, para el caso de parámetros de entrada cuyos valores fueran numéricos, lo que se ha hecho es determinar el rango [a, b] de posibles valores, y crear casos de prueba con al menos los valores: a, b, a – 1, b + 1, y n ϵ (a, b), incluyendo (si es posible) valores positivos, negativos y el cero.

Para la creación de casos de prueba, existen ya diversas metodologías para lograr este objetivo. Concretamente, aquí se ha decidido utilizar pruebas de la *estructura de control*. Se trata de un conjunto de técnicas de prueba de 'caja blanca', que buscan probar todas las posibles combinaciones de datos a través todos los posibles recorridos del flujo de ejecución del código. Está compuesta por las pruebas de la ruta base, pruebas de condiciones, pruebas de flujo de datos y pruebas de bucles.

La *prueba de la ruta base* permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y utilizar esta medida para definir un conjunto base de rutas de ejecución. Los casos de prueba generados para probar este conjunto base nos garantizan que cada instrucción del programa se ejecuta al menos una vez durante las pruebas. Este método hace uso de la notación del 'grafo de flujo' para representar el flujo de control lógico, a partir del cual se puede determinar el número de rutas de ejecución posibles de nuestro programa o 'complejidad ciclomática'. Este valor es, igualmente, el número de casos de prueba que habrá que generar.

Esta técnica de la ruta base es una prueba sencilla y efectiva, pero no suficiente por sí sola. Por ello, se aplican las pruebas descritas a continuación que sirven para complementar lo aportado por ella.

La *prueba de las condiciones* es un método de diseño de casos de prueba que comprueba las condiciones incluidas en un módulo de programa. En una condición simple se compara (==, !=) el resultado de dos expresiones aritméticas (+, -, *, /) mientras que en una condición compuesta aparecen dos o más condiciones simples (con los operadores &&, ||, !). Una condición es incorrecta porque alguno de sus componentes los es, ya sean los operadores booleanos, relacionales o aritméticos, los operandos o los paréntesis. El método de la prueba de las condiciones se centra en probar cada condición del programa para asegurar que no contiene errores.

El método de la *prueba del flujo de datos* selecciona rutas de prueba de un programa considerando dónde se localizan la definición y el uso de una variable determinada en un programa. De este modo, para cada variable del programa se determina su cadena de 'definición-uso', (es decir, la instrucción en que se define, y aquellas en donde es usada), y la estrategia se basa en crear casos de prueba que aseguren que cada cadena se recorre al menos una vez.

La *prueba de los bucles* se centra en la validez de los bucles creados. Existen cuatro clases de bucles (simples, anidados, concatenados y no estructurados) para cada una de las cuales se aplican pruebas de manera distinta. La idea es crear para cada uno un juego de pruebas lo suficientemente variado como para cubrir un número representativo de diferentes pasadas sobre los bucles.

La combinación de los casos de prueba que se obtienen con la aplicación de cada uno de estos cuatro métodos nos da como resultado un conjunto de casos de prueba que logra ejecutar todas las líneas de código desde diferentes perspectivas y cubriendo todos los rangos de valores posibles.

No obstante, hay que aclarar que, para la selección del conjunto final de casos de prueba, se han descartado amplios rangos de estos y se han escogido aquellos que se consideraban más representativos. Esto es debido a que, ya de por sí, cada una de estas metodologías devuelve un número bastante amplio de casos de prueba (si bien muchos de los casos de diferentes metodologías se solapaban, y por tanto, se podían eliminar). Pero al combinar todos los casos de prueba, sobre todo para el caso de la prueba de los bucles, donde los diferentes tipos de bucles ofrecen enormes cantidades de juegos de prueba, el número total de casos de prueba resultaba excesivo y, a fin de completar las pruebas en un tiempo razonable, se ha optado por clasificar los casos de prueba por orden de importancia (basándose en el número de circunstancias posibles que tuvieran en cuenta, y su capacidad de no solaparse con otros casos de prueba) y seleccionar sólo los más representativos, conformando un número final de casos manejable.

Con todo esto, se ha alcanzado un nivel aceptable de corrección en el código creado, en el sentido de que no existen fallos de programación, es decir, que lo que se lleva a cabo, se hace de manera correcta. Ahora queda por comprobar si realiza lo esperado.

## 9.2 Fase 2

La segunda fase de pruebas está pensada para llevarse a cabo una vez terminada una versión completamente funcional de la herramienta, y habiendo pasado satisfactoriamente todas las pruebas de la fase anterior. Llegados a este punto, tenemos una herramienta capaz de llevar a cabo una serie de funciones de manera correcta, donde no hay fallos provocados por la incorrección del código.

Ahora el objetivo de esta segunda fase es demostrar que la herramienta puede llevar a cabo las funciones que se espera de ella, haciendo uso además de un sistema Hydra real, aunque minimalista, es decir, que dicho sistema sólo tendrá que controlar una cantidad muy reducida de nodos. Además, no se le dará importancia a la gestión directa que

realice sobre Hydra, lo que significa que nos centraremos en el desempeño de la herramienta, y no en cómo esto afecta al comportamiento en Hydra. Esto último viene determinado por la complejidad y corrección de la estrategia de balanceo descrita en el archivo de políticas, y requiere de un entorno de pruebas más complejo para su evaluación (éste es el objetivo de la última fase de pruebas).

Así, puesto que la intención es corroborar que la herramienta cumple con las funciones para las que se diseñó, las pruebas que se van a realizar aquí son, en su mayoría, aunque no todas, de tipo 'caja negra', donde se van a utilizar como referencia los requisitos expresados en la sección de 'Especificación de requisitos' de este mismo documento.

Para diseñar los casos de prueba con los cuales se podrá probar que la aplicación es capaz de realizar sus funciones, se van a tener en cuenta tres variables principalmente: las entradas al sistema, el estado actual de éste y las comunicaciones. Estos tres son los puntos críticos del sistema, cuyos posibles valores y sus combinaciones son los que determinan cómo se comporta en cada ciclo de su ejecución y son la fuente de posibles errores inesperados o inevitables (como es el caso de las comunicaciones, que al ser un sistema distribuido podrían fallar en cualquier momento). Hay que tener en cuenta que no sólo se pretende comprobar que el software funciona bien cuando los datos son correctos, sino igualmente cuando son incorrectos, de modo que sea capaz de mostrar los errores o problemas encontrados tanto por consola como por el archivo de registro, como ya se estableció en los requisitos.

### 9.2.1 Entradas al sistema

Las posibles entradas al sistema se describieron en la sección de requisitos, que son el archivo de configuración, y el archivo de definición de las políticas. Se recuerdan aquí las circunstancias posibles que se pueden dar en ambas entradas.

#### *Archivo de configuración*

Circunstancias posibles:

- No existe el archivo.
- El archivo está vacío.
- Contiene las variables correctas que posteriormente serán utilizadas, junto con valores incorrectos.
- Contiene variables incorrectas, es decir, que posteriormente no serán utilizadas durante la ejecución del programa.
- La definición de alguna variable no es correcta:
    - no se usa el separador '=';
    - hay más de una definición en una misma línea;
    - hay variables repetidas.

#### *Archivo de políticas*

Circunstancias posibles:

- No existe el archivo.
- El archivo está vacío.
- Falta la definición explícita de una o ambas secciones del archivo (balanceadores y aplicaciones).
- La estructura del archivo, en forma de árbol, no es la esperada:
    - Alguna palabra reservada está mal escrita.
    - Algún elemento no aparece donde debería.
    - Hay algún elemento de más de lo que se espera recibir.
- En el archivo se definen más aplicaciones que las que está manejando Hydra.
- En el archivo se definen menos aplicaciones que las que está manejando Hydra.
- En el archivo no se definen todos los balanceadores que son referenciados en la sección de aplicaciones.
- En el archivo se definen más balanceadores de los que son referenciados en la sección de aplicaciones.
- La definición de la expresión lógica/matemática de alguna de las condiciones de los estados de las aplicaciones es incorrecta, según el formato que determina la biblioteca *JEval*.
- En la condiciones, aparece alguna variable cuya operación no se reconoce entre el conjunto de posibles operaciones a realizar sobre las variables de monitorización.
- En las condiciones, aparece alguna variable cuya variable de monitorización no aparece entre las que ofrece Hydra sobre las aplicaciones.
- El tipo de datos de una variable de monitorización que aparece en una variable no coincide con lo que espera recibir la operación a realizar sobre ella.
- Existe una aplicación para la cual no se cumple ninguna de las condiciones de sus estados.
- Existe una aplicación para la cual se cumplen dos o más de las condiciones de sus estados.

### 9.2.2 Estado actual del sistema

El estado actual del sistema determinará que, ante una misma entrada, el sistema se comporte de maneras distintas según el estado en que se encuentre en ese momento.

Los factores o elementos sobre los que el sistema guarda información, y cuyas combinaciones, por lo tanto, determinan el estado del sistema, son:

| Elemento | Situación | Descripción |
|---|---|---|
| **1. La configuración del sistema** | 0 | Ya se dispone de datos de configuración. |
| | 1 | Se dispone de nuevos datos de configuración (se ha modificado el anterior archivo de configuración o añadido uno nuevo). |
| **2. Las políticas de control** | 0 | Todavía no hay datos sobre las políticas (no había archivo de políticas o era incorrecto). |
| | 1 | Ya se dispone de datos sobre las políticas. |
| | 2 | Se dispone de nuevos datos sobre las políticas (se ha modificado el anterior archivo de políticas o añadido |

| | | |
|---|---|---|
| | | uno nuevo). |
| **3. El estado actual de cada aplicación que se está analizando** | 0 | No hay ningún estado definido para la aplicación. |
| | 1 | Hay un estado definido para la aplicación, y coincide con el estado de la condición hecha cierta en el ciclo actual. |
| | 2 | Hay un estado definido para la aplicación, y no coincide con el estado de la condición hecha cierta en el ciclo actual. |

Las combinaciones de las situaciones de estos tres elementos determinan el estado en que se encuentra el sistema. Hay que tener en cuenta que estos estados no tienen nada que ver con los estados que se podrían obtener mediante un análisis como si de una máquina de estados se tratara (representable en un diagrama de máquina de estados). Estos estados están enfocados únicamente al desarrollo de la actual fase de pruebas para determinar las posibles situaciones en que nos podemos encontrar, y por lo tanto, que debemos tener en cuenta para la creación de unos buenos casos de prueba.

En general, los tres factores son independientes entre sí, por lo que no hace falta tener en cuenta todas las combinaciones posibles. Sólo hay que tener en cuenta tres circunstancias especiales:

1. En el segundo elemento, la situación 2 sólo es alcanzable si el primer elemento se encuentra en situación 1.
2. Además, también en cuanto al segundo elemento, si en un ciclo dado nos encontramos en la situación 2 (habiéndose leído ya un archivo de políticas, se ha leído uno nuevo), esto obliga al tercer elemento a entrar en situación 0 (pues habría que reiniciar los estados de las aplicaciones basándose en las nuevas condiciones del archivo de políticas nuevo).
3. En el tercer elemento, las situaciones 1 y 2 sólo se pueden dar con la situación 1 del segundo elemento (para que haya estados concretados para las aplicaciones, se tiene que disponer de la información sobre las políticas).

Por tanto, los estados del sistema, según los elementos anteriores, quedan definidos de la siguiente manera:

| Estado | Situación | Descripción |
|---|---|---|
| **0** | 0 0 0 | Ya se dispone de unos datos de configuración (los datos por defecto, o leídos de un archivo correctamente formado), pero aún no se tienen datos sobre las políticas (puesto que no había archivo de políticas, o porque estaba mal formado, y aún no se ha podido leer ninguno anterior). En este punto, tampoco se ha determinado ningún estado para ninguna aplicación. |
| **1** | 1 0 0 | Se acaba de leer un archivo nuevo de configuración y se han incorporado sus datos al sistema, pero aún no se tienen datos sobre las políticas (puesto que el archivo nuevo indicado no existía o estaba mal formado, y aún no se ha podido leer ninguno anterior). En este punto, tampoco se ha determinado ningún estado para ninguna aplicación. |
| **2** | 0 1 0 | Ya se dispone de unos datos de configuración (los datos por defecto, o |

| | | |
|---|---|---|
| | | leídos de un archivo correctamente formado). Igualmente, se dispone de la información de las políticas (de un archivo leído anteriormente). En este caso, no se dispone de un estado definido para la aplicación actual que se está analizando. |
| **3** | 1 1 0 | Se acaba de leer un archivo nuevo de configuración y se han incorporado sus datos al sistema, y del mismo modo, se dispone de la información de las políticas (de un archivo leído anteriormente, puesto que actualmente no se ha definido un archivo nuevo de políticas, o éste no existía o era incorrecto, por lo que se sigue utilizando el último del que se disponía). En este caso, no se dispone de un estado definido para la aplicación actual que se está analizando. |
| | 0 2 0 | Estado imposible de alcanzar. |
| **4** | 1 2 0 | Se han leído nuevos datos de configuración, que incluían un nuevo archivo de políticas. Éste se ha leído igualmente de forma correcta, su información se ha incorporado al sistema, y se han borrado los estados de las aplicaciones guardados anteriormente. |
| | 0 0 1 | Estado imposible de alcanzar. |
| | 1 0 1 | Estado imposible de alcanzar. |
| **5** | 0 1 1 | Ya se dispone de unos datos de configuración (los datos por defecto, o leídos de un archivo correctamente formado). Igualmente, se dispone de la información de las políticas (de un archivo leído anteriormente). En este caso, existe ya un estado determinado para la aplicación que se está analizando actualmente, y éste coincide con el estado cuya condición a resultado positiva en el ciclo actual (es decir, se mantiene el estado ya determinado anteriormente). |
| **6** | 1 1 1 | Se acaba de leer un archivo nuevo de configuración y se han incorporado sus datos al sistema, y del mismo modo, se dispone de la información de las políticas (de un archivo leído anteriormente, puesto que actualmente no se ha definido un archivo nuevo de políticas, o éste no existía o era incorrecto, por lo que se sigue utilizando el último del que se disponía). En este caso, existe ya un estado determinado para la aplicación que se está analizando actualmente, y éste coincide con el estado cuya condición a resultado positiva en el ciclo actual (es decir, se mantiene el estado ya determinado anteriormente). |
| | 0 2 1 | Estado imposible de alcanzar. |
| | 1 2 1 | Estado imposible de alcanzar. |
| | 0 0 2 | Estado imposible de alcanzar. |
| | 1 0 2 | Estado imposible de alcanzar. |
| **7** | 0 1 2 | Ya se dispone de unos datos de configuración (los datos por defecto, o leídos de un archivo correctamente formado). Igualmente, se dispone de la información de las políticas (de un archivo leído anteriormente). En este caso, existe ya un estado determinado para la aplicación que se está analizando actualmente, y éste no coincide con el estado cuya condición a resultado positiva en el ciclo actual (es decir, hay que cambiar el estado que había anteriormente determinado para esa aplicación por el nuevo). |
| **8** | 1 1 2 | Se acaba de leer un archivo nuevo de configuración y se han incorporado sus datos al sistema, y del mismo modo, se dispone de la información de las políticas (de un archivo leído anteriormente, puesto que actualmente no se ha definido un archivo nuevo de políticas, o éste no existía o era incorrecto, por lo que se sigue utilizando el último del que se disponía). En este caso, existe ya un estado determinado para la aplicación que se está analizando actualmente, y éste no coincide con el estado cuya condición a resultado positiva en el ciclo actual (es decir, hay que cambiar el estado que había anteriormente determinado |

| | | |
|---|---|---|
| | | para esa aplicación por el nuevo). |
| | 0 2 2 | Estado imposible de alcanzar. |
| | 1 2 2 | Estado imposible de alcanzar. |

Esto nos deja un total de nueve estados posibles que deberemos considerar para la creación de los casos de uso.

### 9.2.3 Comunicaciones

Las comunicaciones se podrían considerar un apartado especial dentro de esta sección de pruebas, y son muy importantes puesto que son un punto crítico que puede fallar en cualquier momento y que no se puede evitar. Para poder probar apropiadamente esta parte, se necesita un sistema Hydra en ejecución, por eso esta parte no se pudo probar en la anterior fase de pruebas. Lo más importante con referencia a este punto es que el software sea capaz de comunicarse con Hydra correctamente cuando sea necesario, y que cuando falle la comunicación, sea capaz de responder adecuadamente como se previó (mostrar el fallo en el registro, continuar la ejecución y saltar a un nuevo ciclo de ejecución, etc.).

### 9.2.4 Casos de prueba

Mediante la combinación de los diferentes valores o circunstancias que se pueden dar en las entradas al sistema, los estados definidos y la comunicación con Hydra, se ha creado un conjunto de casos de prueba que asegura que las funciones requeridas al sistema se satisfacen en todas las posibles situaciones en que el software podría encontrarse en un momento dado.

De este modo, una vez completada la segunda fase de pruebas, sabemos que el software es capaz de cumplir con los requisitos funcionales para los que se concibió, y es capaz de hacerlo con total corrección. Por último, queda probar la herramienta en un entorno real.

## 9.3 Fase 3

Para la tercera fase de pruebas, se ha querido probar el sistema en un entorno real. El objetivo es, por un lado, asegurar que el software es capaz de hacer lo que debe en un sistema cloud completo y cambiante gestionado por el sistema Hydra, y por otro lado, comprobar que el software satisface las necesidades para las cuales se ideó. Es decir, que cumple con los requisitos que se definieron al principio, y que además estos requisitos se definieron bien y el producto final es capaz de mejorar y ampliar las prestaciones del sistema Hydra como inicialmente se pretendía.

Para estas pruebas, se ha decidido utilizar un entorno cloud público, concretamente los servicios ofrecidos por Amazon Web Services [22] y más específicamente, el servicio Amazon EC2, ‘un servicio web que proporciona capacidad informática con tamaño modificable en la nube, diseñado para facilitar a los desarrolladores *la informática en la nube* escalable basada en web’. Amazon dispone de una modalidad de contratación, por la cual, durante el primer año, y con una serie de restricciones en su uso, se puede

disponer de su sistema cloud sin coste alguno. Las restricciones que establece se refieren a una cantidad limitada en el uso de instancias usables, memoria no volátil, horas de uso totales, etc. Pero todas estas restricciones son perfectamente asumibles para poder llevar a cabo las pruebas que aquí nos planteamos.

El entorno en que se van a realizar las pruebas, por tanto, es en una red privada (virtual) accesible desde el exterior, en la cual podremos lanzar instancias de nodos que serán sistemas Ubuntu 14.04. Para las pruebas se han utilizado un total de diez nodos, de los cuales, uno es el principal, en el que está instalado y ejecutándose tanto el sistema Hydra como nuestra herramienta de control, mientras que el resto de nodos sólo sirven para ejecutar la aplicación que se esté monitorizando, y en ellos sólo hay sondas que recogen la información del nodo y la mandan al principal. La aplicación en ejecución sobre la que Hydra realiza su monitorización y gestión es el servicio 'ssh'.

En realidad, la aplicación elegida es una cuestión secundaria, dado que lo que nos interesa es la monitorización del estado de los nodos (carga de la memoria, la CPU, las conexiones, etc.). Los cambios en estos valores se van a simular mediante el uso de la herramienta 'stress' para Linux. Esta herramienta permite crear una carga de trabajo para los elementos antes mencionados, de modo que se pueden cambiar sus valores sin necesidad de ejecutar aplicaciones pesadas reales.

Lo que se pretende conseguir con las pruebas es lo siguiente: las políticas de balanceo determinan una serie de balanceadores, que a su vez determinan cómo se comportará Hydra a la hora de priorizar unos nodos u otros a los que enviar carga de trabajo, basándose en los valores de las variables de monitorización que se esté controlando en los nodos donde se está ejecutando la aplicación que gestionar (en este caso, ssh, ejecutada por todos los nodos). Hydra se dedicará a gestionar y balancear la carga de trabajo entre sus nodos basándose en la política (balanceadores) que tenga definida en un momento dado, mientras que nuestra aplicación se asegurará periódicamente de que dicha situación no haya cambiado. Mediante el servicio stress, podremos provocar un cambio sustancial a nivel de todo el cloud, que debe ser detectado por la herramienta, lo cual hará que cambie la política de balanceo de Hydra. Así, a continuación se debería poder comprobar que la priorización que realiza Hydra es sustancialmente distinta, basada en la nueva estrategia que se le ha establecido (a través de la definición de nuevos balanceadores).

Finalmente, la ejecución de esta prueba ha durado el equivalente a diez ciclos de ejecución, en cada uno de los cuales se ha extraído información del sistema Hydra para poder rellenar los datos que forman las dos figuras que se pueden ver a continuación. Ambas muestran de una forma visual cómo se ha comportado el sistema y los cambios que se han producido en éste durante el transcurso de toda la prueba.

En el archivo de definición de las políticas de balanceo, se han descrito dos posibles estados en los que se podría encontrar el sistema para la aplicación ssh. Estos dos estados basan su definición en el uso medio de la memoria (*mean_memLoad*), y la

desviación típica en el uso de la CPU (*stdev_cpuLoad*) en los nodos que componían el sistema. La definición exacta es la siguiente:

```
State1
Condition: #{mean_memLoad} <  40 || #{stdev_cpuLoad} <= 10
State2
Condition: #{mean_memLoad} >= 40 && #{stdev_cpuLoad} >  10
```

Para la prueba, lo que nos interesa son los valores de las operaciones mencionadas (media y desviación típica) sobre los valores de las variables de monitorización que en este caso nos interesan (la memoria y la CPU) pues son las que se usan en la definición de las políticas. Así, la Figura 6 muestra la evolución de estos valores a lo largo de los diez ciclos de que se compone la prueba.

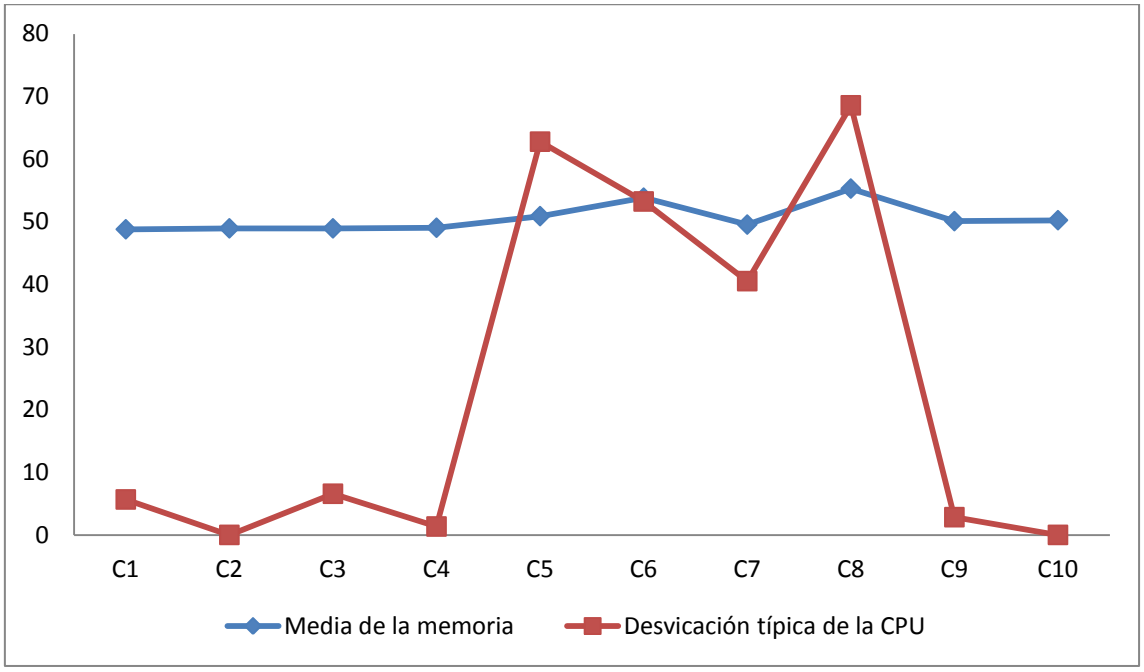


**Figura 6, Evolución de las variables monitorizadas**

Como se puede ver, el uso medio de la memoria apenas varía en el tiempo. Sin embargo, la desviación típica de la CPU sí varía significativamente. Se pueden diferenciar dos partes principales:

1. Un bloque que engloba los cuatro primeros ciclos y los dos últimos, donde dicha desviación típica se mantiene por debajo de 10;
2. Y un segundo bloque, formado por los ciclos 5, 6, 7 y 8, caracterizados por una desviación típica mucho más elevada, superior a 10.

Esa diferencia en cuanto a los valores ha sido el resultado del uso de la aplicación stress en varios de los nodos que componen el sistema distribuido. En ellos, se simuló un gran aumento de la carga de procesamiento durante el tiempo que duraron los cuatro ciclos mencionados.

La Figura 7 muestra, en función de los mismos ciclos que la gráfica anterior, el estado determinado por parte de la aplicación gestora de Hydra para la aplicación ssh basándose en los valores anteriores. Como se puede ver, la aplicación ha sido capaz de diferenciar los dos bloques de valores (igual que se puede hacer visualmente con la ayuda de la figura 6), determinar que corresponden a estados diferentes y cambiar la configuración de los balanceadores de Hydra en consecuencia.

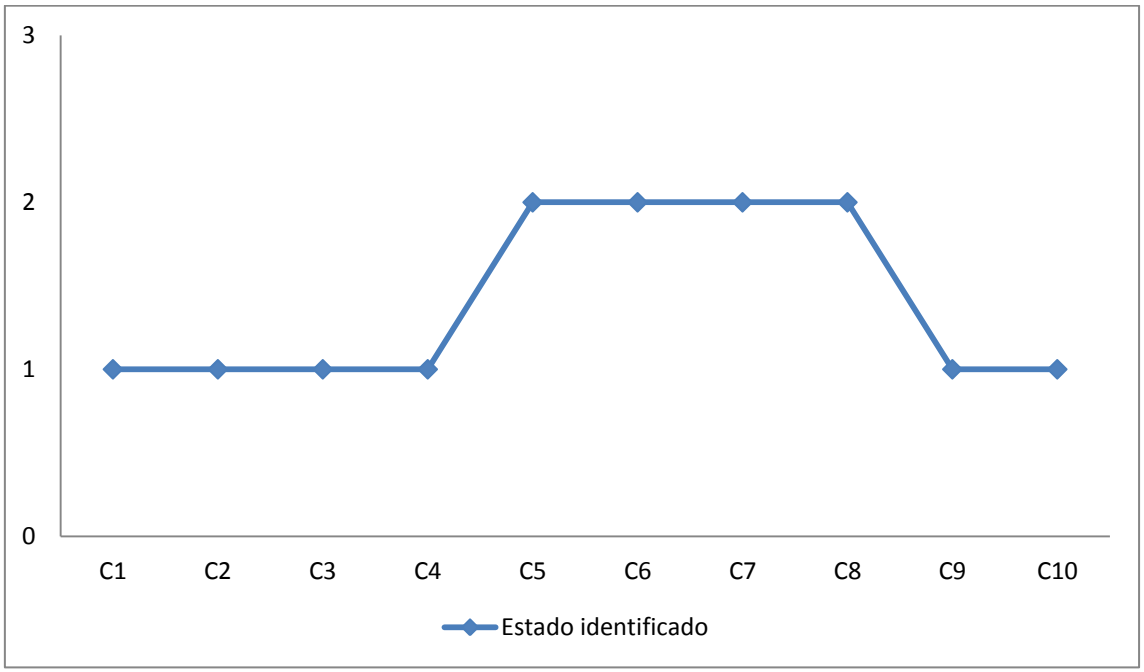


**Figura 7, Estado del sistema**

Del mismo modo, en cada uno de los ciclos de ejecución, se solicitó el orden de los nodos en que Hydra priorizaba para realizar el reparto de la carga de trabajo, y este orden era coherente con los dos estados definidos y los balanceadores que se habían asignado a cada estado, cada uno de los cuales determinaba una priorización distinta de los nodos.

Así pues, tras la finalización satisfactoria de esta última fase de pruebas, se puede decir que el software cumple con los requisitos para los que se diseñó y lo hace de manera correcta (verificado), y además, se ha conseguido el objetivo de mejorar el desempeño del sistema Hydra, mejorando y ampliando sus capacidades de balanceo de carga (validado).

*Parte III*

# Conclusiones y líneas futuras

# 10 Conclusiones

Al comienzo del desarrollo del proyecto, se plantearon como objetivos aquellos que ya se han mencionado en esta memoria: crear una herramienta software que fuera capaz de monitorizar y gestionar la actividad de una herramienta de gestión de entornos distribuidos (Hydra), para que su estrategia de balanceo de carga se adecuara al modelo creado por la metodología de análisis GloBeM, tras estudiar el entorno distribuido en cuestión.

Hydra es un sistema muy moderno, pero aún con poco recorrido. Aunque resulta muy innovador en algunos aspectos, en otros carece de suficiente madurez. Y éste era el caso especialmente del balanceo de carga, donde las estrategias aplicables podían pecar de simplistas en exceso, e incluso de desfasadas.

Pero ahora, con el añadido que supone el uso de esta herramienta y el sistema GloBeM, Hydra puede dar un salto cualitativo, al mejorar en uno de los aspectos más complejos y discutidos de la gestión de los sistemas distribuidos (el balanceo de carga), lo que, con el tiempo, la puede convertir en una herramienta de referencia en cuanto a software de libre distribución de gestión de entornos grid.

Y esto, que a priori parecía una tarea relativamente sencilla (al fin y al cabo, es una herramienta 'intermediaria' entre dos sistemas ya construidos) me ha supuesto a mí, el autor de este PFM, un trabajo descomunal en materia de investigación y aprendizaje sobre las tecnologías aquí tratadas.

A nivel personal, el motivo por el que elegí este proyecto es porque versaba sobre una de las aplicaciones de las tecnologías de la información y la comunicación que más interés han despertado en mí durante mis estudios de grado y, especialmente, del máster al que da conclusión este PFM: los sistemas distribuidos, y más concretamente, el cloud computing.

Así, la primera necesidad a cubrir fue la de conseguir un entendimiento más profundo de estos sistemas, algo que resultaba prioritario para poder entender el funcionamiento de herramientas como Hydra. Y en cuanto a ésta, conocer todo su funcionamiento interno hasta el más mínimo detalle supuso un esfuerzo mucho mayor del esperado, que requirió de más de una entrevista con los desarrolladores de este sistema.

E igualmente, se tuvieron que estudiar otras opciones que al final se desecharon. Tal es el caso de OpenStack [23], una herramienta de código abierto que permite la creación de clústeres de computadores. En un principio, se pensó en este sistema para poder crear el entorno grid que sería gestionado por Hydra, para así poder completar alguna de las fases de pruebas que lo requerían. Pero, tras mucho tiempo para entender la instalación, configuración y funcionamiento de este servicio, se optó por usar la opción de Amazon Web Services. Aunque esto, sin duda, me ha reportado experiencia en cuanto a lo que supone elegir las herramientas a usar en un proyecto, desgraciadamente hizo que el tiempo del proyecto se alargara bastante.

Además de esto, cabe mencionar igualmente el aprendizaje del sistema GloBeM, si bien no para crearlo por mí mismo, pero sí para poder entender cómo las políticas de balanceo de éste se debían aplicar al sistema Hydra.

Sin embargo, todo esto suponía sólo el conocimiento teórico o de las herramientas que después permitirían la construcción del software, pero implementarlo ha sido otra historia. A lo largo del proyecto, he realizado un intenso esfuerzo activo por respetar una metodología estándar para cubrir el ciclo de vida del software. Y es esto lo que más contento me hace estar del resultado final de la herramienta, mucho más completa, en lo que podríamos llamar 'requisitos secundarios', de lo que en principio se planteó, siendo muy completa en su funcionalidad, con un diseño muy orientado a poder ser ampliada en un futuro, y minuciosamente testeada. Y para conseguir esto, he tenido que lidiar con los procesos del ciclo de vida del software y los estándares actuales que les hacen referencia, más de lo que he tenido que hacerlo en cualquier otro momento de mi vida, tanto laboral como estudiantil.

Con todo esto, mi valoración final es muy positiva, sobre todo por la amplitud de distintos conocimientos que he adquirido al tener que investigar sobre una variedad tan grande de herramientas, metodologías, estándares, etc., todas ellas necesarias para completar aquella que, en un principio, 'parecía una herramienta sencilla referida a un tema interesante'.

Y ya que no es oro todo lo que reluce, se hace necesario recalcar el principal punto negativo, que se puede dilucidar de lo ya dicho: la realización del proyecto se ha alargado mucho más de lo deseable. Tanta investigación por parte del autor de tantas tecnologías diferentes, ha resultado en demasiado tiempo de dedicación, y en ocasiones, en bastante estrés. Además, Hydra ha sido, especialmente al principio, una fuente interminable de problemas y frustración. Usar lo moderno e innovador tiene sus ventajas, pero también sus inconvenientes al tratarse de una herramienta inmadura. Para entender esto, resulta muy visual saber que, al comienzo del proyecto, Hydra se encontraba en su versión 1.0, y al final, ésta se encuentra en la 3.2. Las adaptaciones fueron necesarias y agotadoras.

Con todo esto, resulta muy reconfortante haber podido alcanzar con éxito los objetivos propuestos, consiguiendo un software de un rendimiento muy aceptable y muy completo en todo su diseño. Y tal es así, que ya se encuentra a disposición del público en general a través de la página github.com, el mismo repositorio donde está la herramienta Hydra.

Y es que este software se creó, además, con la idea de ofrecerse como código abierto, apostando por apoyar el sistema Hydra. Así pues, el verdadero éxito de este proyecto está en el atractivo que consiga generar para la comunidad de desarrolladores libres, que decidan utilizar y seguir ampliando las capacidades que esta herramienta ofrece.

Pues aún hay trabajo que hacer…

# 11 Líneas futuras

El resultado final de la herramienta, da pie a que se realice trabajo de mantenimiento y evolución de ésta. Y no sólo lo evidente, como es el adaptarla a posibles versiones futuras de Hydra. Por lo menos, se pueden resaltar dos.

Por un lado, el sistema GloBeM puede usar una variedad muy amplia de medidores y descriptores estadísticos a la hora de analizar y describir el modelo del sistema grid en cuestión. Aun así, aquí sólo se han tenido en cuenta cuatro de los más básicos: la media aritmética, la desviación típica y los valores máximo y mínimo. En este sentido, añadir el código necesario para ejecutar más operaciones estadísticas ampliaría las posibilidades de aplicación del método GloBeM.

Y por otro lado, la herramienta ha quedado con una carencia que ya fue mencionada a lo largo de este documento. Se trata de la imposibilidad de realizar operaciones de tipo ‘transacción’ durante la gestión de la configuración de Hydra. Resultaría enormemente beneficioso que esta herramienta se adaptara a este requisito, al tiempo que en un posible futuro se incluyera esta opción en el funcionamiento de Hydra.

# 12 Referencias

## 12.1 Referencias


[1] Jesús Montes Sánchez, Alberto Sánchez Campos, y María S. Pérez, "GloBeM: Un modelo global del grid", XX Jornadas de Paralelismo, A Coruña, 16-18 de septiembre de 2009, pp. 617-622

[2] www.laredinfinita.wordpress.com/2014/05/11/balanceo-de-carga/, 14 de enero de 2015

[3] www.computerworld.es/tendencias/que-es-el-balanceo-de-carga, 14 de enero de 2015

[4] www.ecured.cu/index.php/Balanceo_de_cargas, 14 de enero de 2015

[5] www.github.com/innotech/, 12 de enero de 2015

[6] www.unirioja.es/cu/fgarcia/sd/pub/teo/01-IntroduccionALaComputacionDistribuida.pdf, 15 de enero de 2015

[7] www.en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/Density-Based_Clustering, 18 de enero de 2015

[8] www.cis.temple.edu/~giorgio/cis587/readings/id3-c45.html, 25 de noviembre de 2014

[9] www.github.com/coreos/etcd, 12 de enero de 2015

[10] www.thenewstack.io/about-etcd-the-distributed-key-value-store-used-for-kubernetes-googles-cluster-container-manager/, 3 de febrero de 2015

[11] www.coreos.com/using-coreos/etcd/, 3 de febrero de 2015

[12] Institute of Electrical and Electronics Engineers, 'IEEE Std. 830-1998, IEEE Recommended Practice for Software Requirements Specifications', E-ISBN: 978-0-7381-0448-5, 20 October 1998

[13] http://json.org/, 13 de enero de 2015

[14] http://jeval.sourceforge.net/, 12 de enero de 2015

[15] http://jeval.sourceforge.net/docs/api/net/sourceforge/jeval/Evaluator.html, 12 de enero de 2015

[16] Institute of Electrical and Electronics Engineers, 'IEEE Std. 1016-2009, IEEE Standard for Information Technology – Systems Design – Software Design Descriptions', E-ISBN: 978-0-7381-5925-6, 20 July 2009

[17] www.uml-diagrams.org/, 8 de enero de 2015

[18] www.omg.org/gettingstarted/what_is_uml.htm, 8 de enero de 2015

[19] www.sparxsystems.com.ar/resources/tutorial/uml2_compositediagram.html, 8 de enero de 2015

[20] https://www.modelio.org/about-modelio/features.html, 2 de febrero de 2015

[21] https://netbeans.org/features/index.html, 2 de febrero de 2015

[22] http://aws.amazon.com/es/about-aws/, 10 de febrero de 2015

[23] https://www.openstack.org/, 22 de enero de 2015

[24] https://www.java.com/es/about/whatis_java.jsp, 2 de febrero de 2015

## 12.2 Artículos científicos relacionados

Gabriel Antoniu, Bunjamin Memishi, Jesús Montes, Alberto Sánchez and María S. Pérez, "GMonE: A complete approach to cloud monitoring", The International Journal of Grid Computing and eScience, vol. 29, issue 8, October 2013, pp. 2026-2040

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xi-aowei Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", Proc. of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996, pp. 226-231

Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero, "The grid as a single entity: Towards a behavior model of the whole grid", in OTM Conferences (1), 2008, pp. 886-897

Jesús Montes, Alberto Sánchez, Julio J. Valdés, María S. Pérez, and Pilar Herrero, "Finding order in chaos: a behavior model of the whole grid", Concurrency and Computation: Practice and experience, 16 September 2009

Ross J. Quinlan, "C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)", Morgan Kaufmann, January 1993

# Anexo A, Descripción de la biblioteca JEval

La biblioteca JEval es la elegida para la escritura y evaluación de las expresiones lógicas y matemáticas que se utilizan en las condiciones de los estados de las aplicaciones en el archivo de políticas de balanceo.

Para poder evaluar correctamente las expresiones, éstas deben ceñirse al formato definido por la biblioteca JEval.

Se da soporte para los siguientes operadores:

- ( abrir paréntesis
- ) cerrar paréntesis
- + suma
- - substracción
- * multiplicación
- / división
- % módulo
- + más unitario
- - menos unitario
- = igual que
- != distinto de
- < menor que
- <= menor o igual que
- > mayor que
- >= mayor o igual que
- && conjunción
- || disyunción
- ! negación

Variables:

- Las variables deben estar encerradas con el signo de apertura '#{' y el signo de cierre '}', sin las comillas.
- Se incluyen dos variables predefinidas: E y PI, las cuales representan el mismo valor que las constantes Math.E y Math.PI en la API estándar de Java [24].

Notas sobre el análisis de expresiones:

- Se ignoran los espacios al analizar expresiones.
- El orden de precedencia que se utiliza es el siguiente de mayor a menor.
- La expresión se evalúa como una o más subexpresiones. Las subexpresiones entre paréntesis se evalúan antes que otras partes de la expresión.
- Las subexpresiones más internas se evalúan primero, moviéndose hacia afuera.
- Las subexpresiones del mismo nivel se evalúan de izquierda a derecha.

- Al evaluar expresiones y subexpresiones, los operadores se evalúan con el siguiente orden de prioridad:
    - Los operadores con la misma precedencia se evalúan de izquierda a derecha.
    - Una vez que la expresión es analizada, las variables son reemplazadas por sus valores. El evaluador tiene su propio mapa interno de variables que utiliza para resolver los valores de variables.
    - A continuación, se ejecutan las funciones y se reemplazan con sus resultados. Cada argumento de la función es evaluado individualmente como una subexpresión.
    - Una vez resueltas todas las variables y funciones, entonces la expresión y subexpresiones analizadas son evaluadas de acuerdo a la precedencia de los operadores.

Precedencia de los operadores:

- + más unitario, - menos unitario, ! negación
- multiplicación, / división, % módulo
- + suma, - substracción
- < menor que, <= menor o igual que, > mayor que, >= mayor o igual que
- = igual que, != distinto de
- && conjunción
- || disyunción

Los nombres de funciones y variables no pueden romper las siguientes reglas:

- no pueden comenzar con un número
- no pueden contener un operador
- no pueden contener un carácter de comillas, simples o dobles
- no pueden contener un carácter de llave
- no pueden contener uno de los siguientes caracteres especiales: #, ~, ^, !

## Anexo B, Ejemplo de un fichero de políticas

A continuación, se puede ver un ejemplo del posible contenido del archivo de políticas de balanceo, creado a partir del modelo generado por GloBeM, en su formato JSON correspondiente.

```
[{"Balancers": [
      {"name": "balancer11",
      "parameters": [
            {"name": "worker",
            "value": "SortByNumber"
            },
            {"name": "sortAttr",
            "value": "cpuLoad"
            },
            {"name": "order",
            "value": "1"
            }
      ]},
      {"name": "balancer12",
      "parameters": [
            {"name": "worker",
            "value": "SortByNumber"
            },
            {"name": "sortAttr",
            "value": "connections"
            },
            {"name": "order",
            "value": "0"
            }
      ]},
      {"name": "balancer21",
      "parameters": [
            {"name": "worker",
            "value": "SortByNumber"
            },
            {"name": "sortAttr",
            "value": "memLoad"
            },
            {"name": "order",
            "value": "1"
            }
      ]}
]},
{"Apps": [
      {"name": "hydra",
      "states": [
            {"name": "state1",
            "condition": "#{stdev_cpuLoad} < 1",
            "balancers": "balancer11"
            },
            {"name": "state2",
            "condition": "#{stdev_cpuLoad} == 1",
            "balancers": "balancer12"
            },
            {"name": "state3",
            "condition": "#{stdev_cpuLoad} > 1",
            "balancers": "balancer21"
            }
      ]},
```

```
      {"name": "ssh",
      "states": [
            {"name": "state1",
            "condition": "#{mean_memLoad} < 40 && #{stdev_cpuLoad} <
1",
            "balancers": "balancer11"
            },
            {"name": "state2",
            "condition": "#{mean_memLoad} >= 40 || #{stdev_cpuLoad} >=
1",
            "balancers": "balancer12;balancer21"
            }
      ]},
      {"name": "App3",
      "states": [
            {"name": "state1",
            "condition": "#{max_connections} == 1",
            "balancers": "balancer12"
            },
            {"name": "state2",
            "condition": "#{max_connections} == 2",
            "balancers": "balancer21"
            }
      ]}
]}]
```

## Anexo C, JSON ejemplo devuelto por Hydra

A continuación, se muestra un ejemplo de los datos devueltos por el sistema Hydra al hacer una solicitud de toda la información del sistema grid que está manejando. La información viene agrupada por aplicaciones, que en este caso son 'ssh' e 'hydra'. Y de cada una, muestra los balanceadores que tiene definidos (junto con sus parámetros), y las instancias, es decir, todos los nodos que pueden ejecutar la aplicación, así como sus variables de monitorización.


```
{"action":"get",
 "node":{
      "key":"/db/apps",
      "dir":true,
      "nodes":[{
            "key":"/db/apps/ssh",
            "dir":true,
            "nodes":[{
                  "key":"/db/apps/ssh/Balancers",
                  "dir":true,
                  "nodes":[{
                        "key":"/db/apps/ssh/Balancers/0",
                        "dir":true,
                        "nodes":[{
                              "key":"/db/apps/ssh/Balancers/0/worker",
                              "value":"SortByNumber",
                              "modifiedIndex":2,
                              "createdIndex":2
                        },
                        {
                              "key":"/db/apps/ssh/Balancers/0/sortAttr"
                        ,
                              "value":"cpuLoad",
                              "modifiedIndex":3,
                              "createdIndex":3
                        },
                        {
                              "key":"/db/apps/ssh/Balancers/0/order",
                              "value":"1.00",
                              "modifiedIndex":4,
                              "createdIndex":4
                        }],
                        "modifiedIndex":2,
                        "createdIndex":2
                  }],
                  "modifiedIndex":2,
                  "createdIndex":2
            },
            {
                  "key":"/db/apps/ssh/Instances",
                  "dir":true,
                  "nodes":[{
                        "key":"/db/apps/ssh/Instances/javier-
                  VirtualBox",
                        "dir":true,
                        "nodes":[{
                              "key":"/db/apps/ssh/Instances/hydra0/demo
                        Mode",
                              "value":"unlocked",
```

```
                        "expiration":"2014-12-
                  07T19:29:57.425330693+01:00",
                        "ttl":28,
                        "modifiedIndex":228,
                        "createdIndex":228
                  },
                  {
                        "key":"/db/apps/ssh/Instances/hydra0/stat
                  e",
                        "value":"1.00",
                        "expiration":"2014-12-
                  07T19:29:57.425521404+01:00",
                        "ttl":28,
                        "modifiedIndex":229,
                        "createdIndex":229
                  },
                  {
                        "key":"/db/apps/ssh/Instances/hydra0/cost
                  ",
                        "value":"5",
                        "expiration":"2014-12-
                  07T19:29:57.425706251+01:00",
                        "ttl":28,
                        "modifiedIndex":230,
                        "createdIndex":230
                  },
                  {
                        "key":"/db/apps/ssh/Instances/hydra0/uri"
                        ,
                        "value":"ssh Hydra0",
                        "expiration":"2014-12-
                  07T19:29:57.425877696+01:00",
                        "ttl":28,
                        "modifiedIndex":231,
                        "createdIndex":231
                  },
                  {
                        "key":"/db/apps/ssh/Instances/hydra0/clou
                  d",
                        "value":"susecloud",
                        "expiration":"2014-12-
                  07T19:29:57.426046069+01:00",
                        "ttl":28,
                        "modifiedIndex":232,
                        "createdIndex":232
                  }],
                  "modifiedIndex":8,
                  "createdIndex":8
            }],
            "modifiedIndex":8,
            "createdIndex":8
      }],
      "modifiedIndex":2,
      "createdIndex":2
},
{
      "key":"/db/apps/hydra",
      "dir":true,
      "nodes":[{
            "key":"/db/apps/hydra/Balancers",
            "dir":true,
```

```
                  "nodes":[{
                        "key":"/db/apps/hydra/Balancers/0",
                        "dir":true,
                        "nodes":[{
                              "key":"/db/apps/hydra/Balancers/0/worker"
                        ,
                              "value":"SortByNumber",
                              "modifiedIndex":5,
                              "createdIndex":5
                        },
                        {
                              "key":"/db/apps/hydra/Balancers/0/sortAtt
                        r",
                              "value":"cpuLoad",
                              "modifiedIndex":6,
                              "createdIndex":6
                        },
                        {
                              "key":"/db/apps/hydra/Balancers/0/order",
                              "value":"1.00",
                              "modifiedIndex":7,
                              "createdIndex":7
                        }],
                        "modifiedIndex":5,
                        "createdIndex":5
                  }],
                  "modifiedIndex":5,
                  "createdIndex":5
            }],
            "modifiedIndex":5,
            "createdIndex":5
      }],
      "modifiedIndex":2,
      "createdIndex":2
}}
```